Machine Learning – Project 2
Recommender System

Natalia Nessler
School of Computer and Communication Sciences, EPFL, Switzerland

***Abstract*—The problem of recommending items to users is very important these years. Often we don't know anything about the users, but only the previous items they consumed or rated; still, we must find a way to recommend to them something new they will probably like. The goal of this project is to recommend new items to users, based on their previous ratings and using machine learning techniques. The result is then submitted to the competition arena[1] and has shared the second place with some other teams, with RMSE 1.017.**

I. Introduction

This paper describes our work on the project 2 of the Machine Learning course. In particular, we explore the provided data in Section II and go through all the methods we implemented in Sections III and IV. Our work is reproducible by a reader having a similar dataset. Finally, in Section V we discuss the results of our work. The Section VI suggests further improvements.

II. The Data

A. Structure

In this project, 10000 users rate 1000 items, for example movies, from 1 to 5. The data consists of these ratings, given by users to movies, and the goal is to predict new ratings.

The training set is a *1176952 x 2* matrix, where the first column consists of pairs of users and movies and the second column consists of ratings. The entries of the first column are in the following format: *rXcY*, where *X* and *Y* are integers; this means that a user *X* has rated an item *Y*. The item in the second column and the same row represents the corresponding rating.

The test set is a matrix in the same format. It contains the pairs of users and movies for which we must make a prediction of a rating.

B. Pre-processing

1) Sparse matrix: We rearrange the data in a 10000x1000 sparse matrix, where the first row contains the movies indices, the first column contains the users indices, and all other cells contain the rating given by the corresponding user to the corresponding movie. This matrix is sparse because most of the users rarely rate movies. The most rated movie has 4590 rates, and the most active user rated 522 movies; but in average, a user rates 118 movies and a movie is rated 1177 times. Unfortunately, some users have extremely few ratings, but we still must make a prediction for them.

[1]https://www.aicrowd.com/challenges/epfl-ml-recommender-system-2019/leaderboards

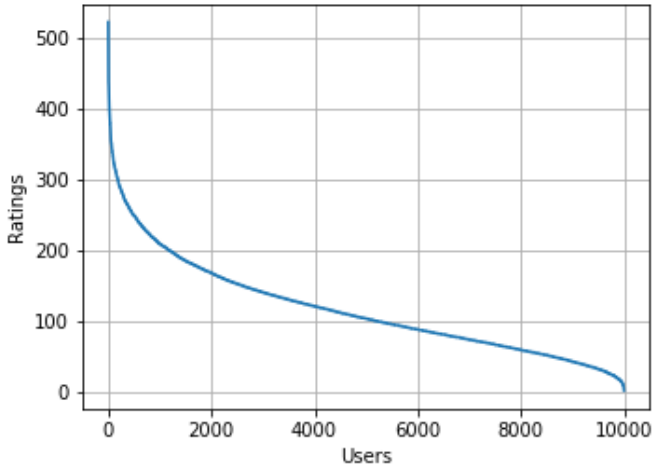

Fig. 1. Ratings per users

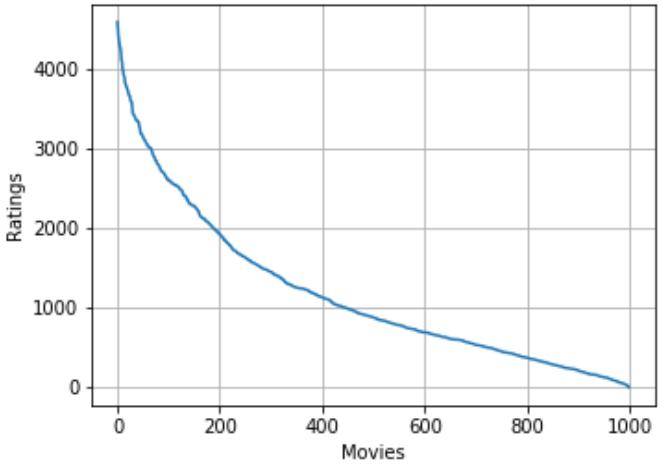

Fig. 2. Ratings per movies

2) Data split: In order to test our model locally before submitting it to the competition arena, we divide the training set in two sets: one for training and the other one for testing. By default, the ratio is the following: 90% of data becomes the new training set and 10% of remaining data becomes the new test set.

C. Evaluation

The prediction is evaluated by measuring the distance to the test set, using the root-mean-squared error. Our goal is to

minimize it. When not specified, the RMSE in this report is the result of local testing rather than the result of submission to the competition arena.

III. Models

We made use of several external libraries, in particular **Surprise** and **sklearn**. The following models have been implemented throughout the project, with the best parameters identified by Grid Search:

A. Baseline

This is a simple model, taking into account the user and movie biases. The rating predicted for the user *u* and the item *i* becomes:

$r_{ui} = \mu + b_u + b_i$,

where μ is the global mean, b_u is the user bias, and b_i is the item bias.

This model worked surprisingly well, with RMSE 0.9978.

B. Basic KNN

This is a basic collaborative algorithm taking into account the K nearest neighbours. There are two versions:

1) User-based: $r_{ui} = \frac{f_1}{f_2}$, where

$\sum_{v \in N_i^k(u)} sim(u,v) \cdot r_{vi}$

$\sum_{v \in N_i^k(u)} sim(u,v)$

and *sim()* is the similarity function computed using Pearson correlation coefficient.

2) Item-based: $r_{ui} = \frac{f_1}{f_2}$, where

$\sum_{j \in N_u^k(i)} sim(i,j) \cdot r_{uj}$

$\sum_{j \in N_u^k(i)} sim(i,j)$

Both user-based and item-based models give the same result with RMSE 1.0235.

C. KNN Baseline

This model takes into account K nearest neighbours and the baseline rating. Two versions of the KNN Baseline are implemented:

1) User-based: $r_{ui} = b_u i + \frac{f_1}{f_2}$, where

$f_1 = \sum_{v \in N_i^k(u)} sim(u,v) \cdot (r_{vi} - b_v i)$

$f_2 = \sum_{v \in N_i^k(u)} sim(u,v)$

2) Item-based: $r_{ui} = b_u i + \frac{f_1}{f_2}$, where

$f_1 = \sum_{j \in N_u^k(i)} sim(i,j) \cdot (r_{uj} - b_u j)$

$f_2 = \sum_{j \in N_u^k(i)} sim(i,j)$

Both versions give the same result with RMSE 1.0064.

D. KNN-means

Similarly to the previous model, this one takes into account the mean rating of users and of movies. There are also two versions:

1) User-based: $r_{ui} = \mu_u + \frac{f_1}{f_2}$, where

$f_1 = \sum_{v \in N_i^k(u)} sim(u,v) \cdot (r_{vi} - \mu_v)$

$f_2 = \sum_{v \in N_i^k(u)} sim(u,v)$

2) Item-based: $r_{ui} = \mu_i + \frac{f_1}{f_2}$, where

$f_1 = \sum_{j \in N_u^k(i)} sim(i,j) \cdot (r_{uj} - \mu_j)$

$f_2 = \sum_{j \in N_u^k(i)} sim(i,j)$

Again, both versions give the same result with RMSE 1.0187.

E. KNN-zscore

Another vesrion of KNN-inspired model, this time taking into account the z-score normalization of each user.

1) User-based: $r_{ui} = \mu_u + \sigma_u \frac{f_1}{f_2}$, where

$f_1 = \sum_{v \in N_i^k(u)} sim(u,v) \cdot (r_{vi} - \mu_v)/\sigma_v$

$f_2 = \sum_{v \in N_i^k(u)} sim(u,v)$

2) Item-based: $r_{ui} = \mu_i + \sigma_i \frac{f_1}{f_2}$, where

$f_1 = \sum_{j \in N_u^k(i)} sim(i,j) \cdot (r_{uj} - \mu_j)/\sigma_j$

$f_2 = \sum_{j \in N_u^k(i)} sim(i,j)$

Both versions give RMSE 1.0206.

F. Slope One

This is a simple collaborative model, focusing on relevant items:

$r_{ui} = \mu_u + \frac{1}{|R_i(u)|} \sum_{j \in R_i(u)} dev(i,j)$,

where $R_i(u)$ are items rated by user *u* and having at least one common user with *i*, and $dev(i,j)$ is the average difference between the ratings of *i* and *j*:

$dev(i,j) = \frac{1}{|U_{ij}|} \sum_{u \in U_{ij}} r_{ui} - r_{uj}$

This model performs well and gives a prediction with RMSE 0.9984.

G. SVD

A matrix decomposition approach may be very interesting for this kind of problem. The SVD algorithm uses biases and minimizes the error using SGD. The prediction is computed as follows:

$r_{ui} = \mu + b_u + b_i + q_i^T p_u$,

where b_u is the user bias, b_i is the item bias, and $q_i^T p_u$ is the remaining factorization.

The SVD method performs well with RMSE 0.9998.

H. SVD++

The SVD++ is a more complicated version of SVD taking into account the implicit ratings, i.e. the fact that a user rated an item, rather than the rating itself.

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

y_j are the item factors of the implicit ratings.

This model performs a little bit better than SVD, but takes significally more time to run. The resulting RMSE is 0.9991.

I. Non-negative matrix factorization

This model is similar to SVD, but the user and item factors are forced to be positive.

$r_{ui} = q_i^T p_u$

The prediction is slightly worse than SVD and SVD++, with RMSE 1.0073.

J. Stochastic Gradient Descent

Here, we do the matrix factorization using the simple Stochastic Gradient Descent. We consider the user features and the movie features separately and update them alternately. It is important to note that, while initializing the feature vectors, we fill the user features uniformly at random, while the movie features are initialized with the average ratings for each movie.

This method gives us a prediction with RMSE 1.0209.

K. Alternating Least Squares

Finally, we implement another matrix factorization algorithm, ALS: it is similar to SGD, but updates the features vectors in a different way.

This model shows the best results with RMSE 0.9847.

IV. Combining models

Now we can compare all the implemented models.

Model	RMSE
ALS	0.9847
Baseline	0.9978
Slope One	0.9984
SVD++	0.9991
SVD	0.9998
KNN Baseline	1.0064
NMF	1.0078
KNN-means	1.0187
KNN-zscore	1.0206
SGD	1.0209
Basic KNN	1.0235

TABLE I
RMSE FOR EACH MODEL WITH THE BEST PARAMETERS.

The best results are shown by ALS model, and Baseline, Slope One, SVD++ and SVD also perform well. We may think of some models as useless, as they show a high RMSE; however, they might capture some particular information not captured by better models, and perform well in some specific cases. It is interesting to combine all our models to make use of all their advantages and capture as much information as possible.

To do this, we create a new *1176952 x 11* matrix, concatenating all the predictions, one prediction per column. Now we need to find the weights for each model in such a way that their weighted sum brings us as near as possible to the test set.

A. Ridge regression

To do so, we use Ridge Regression and obtain the weights for each model as in the Table II. As expected, the ALS and Baseline models have the most important weights.

Now, we use these weights to combine our final predictions required by the competition arena. This method makes a great difference and brings us to the top of the ranking with RMSE 1.019 on the AIcrowd.

Model	Weight
ALS	1.2126
Baseline	-1.2074
Slope One	0.5159
SVD++	0.4634
SVD	0.0718
KNN Baseline	-0.1535
NMF	-0.0242
KNN-means	-0.5221
KNN-zscore	0.4379
SGD	-0.0102
Basic KNN	0.2527

TABLE II
WEIGHT FOR EACH MODEL.

B. Neural network

We also implement a neural network using **Keras**. This method also gives us good results, but not as good as the Ridge Regression. The RMSE on AIcrowd is 1.025.

C. Polynomial expansion

Finally, we overcome the linearity issue and extend the features using a polynomial basis. For this, we multiply the features between them up to a certain degree. The degree 3 shows the best results, and the RMSE on AIcrowd decreases to 1.017, which is our best result.

V. Results

We tested all our models with different combinations of parameters; we then tested different ways of combining them. The following table shows our final results.

Model	RMSE	AIcrowd
Ridge regression with polynomial expansion		1.017
Neural network with polynomial expansion		1.018
Ridge regression combining all models		1.019
Neural network combining all models		1.025
ALS	0.9847	1.028
Baseline	0.9978	1.040
Slope One	0.9984	
SVD++	0.9991	
SVD	0.9998	
KNN Baseline	1.0064	
NMF	1.0078	
KNN-means	1.0187	
KNN-zscore	1.0206	
SGD	1.0209	
Basic KNN	1.0235	

TABLE III
LOCAL RMSE OR AICROWD RMSE FOR EACH MODEL

The best model has been a combination of all implemented models, with ridge regression on their polynomial expansion of degree 3.

VI. Further Work

The work on this project being limited in time, there is still a lot of improvements to do.

A. Model selection

First of all, not all models bring advantages to the overall combination. It would be interesting to test several combinations of models and see whether we should keep them all. This would show us which models are really relevant and certainly improve our results.

B. Refining the parameters

For technical reasons, we couldn't test all possible parameters for the models because some of them took too much time to run. Instead, we did a lot of research to properly chose a small set of parameters to test. With more time and more computational power, we could refine our parameters to get even better results.

VII. Summary

Throughout this project, we have implemented an important amount of models. The best method turned out to be a combination of all the models using the polynomial feature extension. With this method, we've predicted the ratings a user would give to a movie, and obtained a high RMSE of 1.017 on the competition arena.