Machine Learning: CS-433 Class Project 2
Recommender System

Team: ***RoadRunners***
Asli Yorusun, asli.yorusun@epfl.ch
Erdem Bocugoz, erdem.bocugoz@epfl.ch
Serif Soner Serbest, serif.serbest@epfl.ch

School of Computer and Communication Sciences, EPF Lausanne, Switzerland

***Abstract*—This report contains details of our work in the second class project, that we chose *Project: Recommender System* among other proposed projects, of the Machine Learning course, taught at EPFL. The task is to develop a recommendation system using various collaborative filtering methods to predict good recommendations on the given pseudonymized movies' data of users. In order to have the 1.018 prediction error measured by root-mean-squared error (RMSE) compared with ground truth values, on the *CrowdAI*, ensemble learning approach with collaborative filtering methods is used.**

I. INTRODUCTION

The aim of this project is to predict ratings of the movies according to users to build a recommendations system. Collaborative filtering is a set of techniques that analyze similarities between users to identify new user-item associations during the creation of recommender systems by using the assumption that people who have similar tastes are more likely to rate a new item in a similar manner [1]. The main idea of collaborative filtering based (CF-based) algorithms is to provide item recommendations or predictions based on the user's previous ratings and the opinions of other like-minded users. The opinions of users can be obtained explicitly from the users or by using some implicit measures. In this report, we provide a detailed analysis of CF-based recommender system algorithms on the dataset without any additional information about the movies or users. Firstly, we will perform a data exploration in order to evaluate the data quality. Then, we will explain the models that we used, starting from baseline algorithms to more sophisticated approaches. We will continue with explaining our blending approach that is inspired by *BellKor's Pragmatic Chaos*, winner of 2009 Netflix Prize [2]. Finally, we will provide and discuss the results end of the report.

II. DATA EXPLORATION

A. Data Description

The data is composed of ratings from $10,000$ users on $1,000$ movies, also called items, as integers from 1 to 5. $1,176,952$ ratings are provided as the training set which represent around 11.8% of the data.

B. Users

1) Participation: According to figure (II-B1) , users are satiably active. 75% of the users rate movies more than 500 times and the number of ratings increases exponentially among users. In a similar manner, 66 % of the movies are rated more than 100 times and number of ratings increases up to 500. This shows that participation need of machine learning algorithms is satisfied in the data.

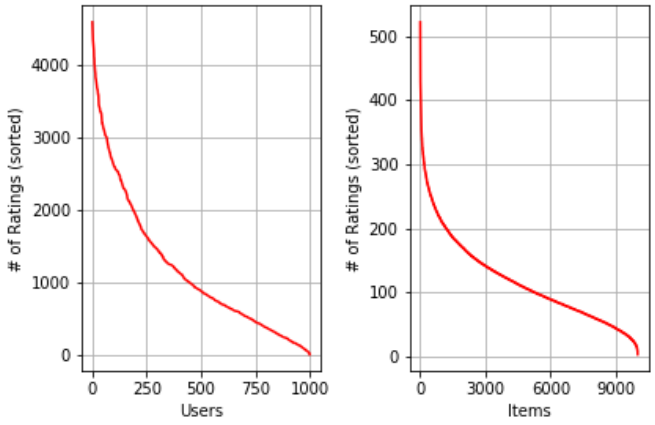

Figure 1. Participation of Users and rating numbers of movies.

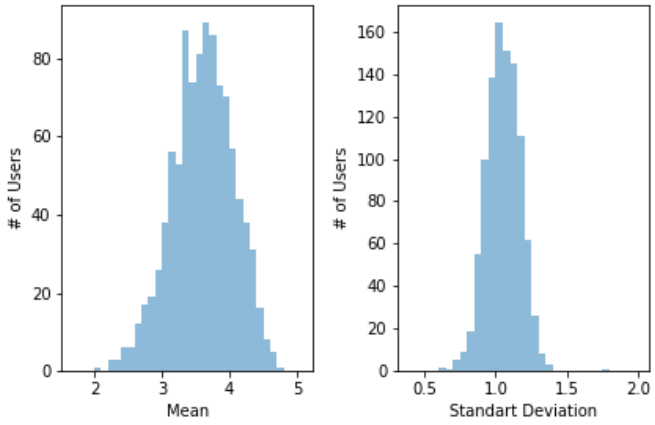

Figure 2. Rating distributions of users.

2) Rating Validity and Consistency: As it is seen in the figure (II-B1), characteristic of the distributions are similar to Gaussian. Hence, data is lack of unrealistic users who may constantly give the same rate to movies or randomly rate the movies.

III. MODELS AND METHODS

A. Global Mean

The simplest model which can also be seen as a baseline is to take the mean of all of the ratings and return the value as the prediction.

B. User/Movie Mean

A better approach is to take the means according to a user or movie and make a prediction specific to the user or the movie. These approaches are used to understand the data and the baseline in the loss function.

C. Baseline

Baseline algorithm predicts the rating by calculating user and movie biases and global mean. The method, called BaselineOnly, is implemented in the Surprise library [3].

$$\widehat{r}_{ui} = \mu + b_u + b_i \tag{1}$$

D. Slope One

Slope One is another fast and simple method [4]. Prediction model sets $\hat{r}_{ui}$ as:

$$\widehat{r}_{ui} = \mu_u + \frac{1}{|R_i(u)|} \sum_{j \in R_i(u)} dev(i,j) \tag{2}$$

where $R_u(i)$ is the set of relevant items, i.e. the set of items j rated by u that also have at least one common user with i, and dev(i,j) is defined as the average difference between the ratings of i and those of j. The method is implemented in the Surprise library.

E. Matrix Factorization with Stochastic Gradient Descent

Given D items, N users and the corresponding rating matrix $X \in R^{D \times N}$, matrix factorization model aims to decompose rating matrix into two lower rank matrices $W \in R^{D \times K}$ and $Z \in R.^{N \times K}$

$$E = \frac{1}{2} \sum_{d,n \in D,N} \left(x_{dn} - \left(W Z^T \right)_{dn} \right)^2 + \frac{\lambda}{2} \|W\|^2 + \frac{\lambda}{2} \|Z\|^2 \tag{3}$$

Stochastic Gradient Descent (SGD) is a faster variant of the Gradient Descent (GD) optimization. While in GD the whole training set is considered before taking one Model Parameters Update step, in SGD only one randomly chosen element is considered for each Model Parameters Update Step, cycling over the training set. SGD is largely preferred over GD since its convergence to a local minimum is almost certain in large data sets. The method is implemented in the PyFM library [5].

F. Matrix Factorization with Alternative Least Squares

Alternative Least Squares (ALS) is another approach, such as SGD, to optimize the equation 3. It is an algorithm that degrades the optimization problem into least squares problem by iteratively fixing one of the two matrices W or Z, and optimizing the other. Simple least squares optimization technique is applied in each iteration.

The method is implemented in PySpark. Apache Spark is an open-source distributed general-purpose cluster computing framework with (mostly) in-memory data processing engine that can do ETL, analytics, machine learning and graph processing on large volumes of data. PySpark is a Python API for Spark that has a machine learning library and functions for recommendation systems.

G. Singular Value Decomposition

Similar to matrix factorization, Singular Value Decomposition (SVD) decomposes the matrix X into W and Z of the given rank which minimizes the sum-squared distance to the target matrix X. The method is implemented in the Surprise library as SVD and an extended version, SVDpp.

H. K-Nearest Neighbors

K-nearest neighbor finds the k most similar items to a particular instance based on a given distance metric like cosine, mean squared difference or pearson. It uses the items, called neighbors, to predict the instance [6].

$$\widehat{r}_{ui} = \frac{\sum\limits_{v \in N_i^k(u)} \text{sim}(u,v) \cdot r_{vi}}{\sum\limits_{v \in N_i^k(u)} \text{sim}(u,v)} \tag{4}$$

A weighted average of neighbors are used to predict the elements according to their similarities as it is seen in equation 4. The method is available in the Surprise library as KNNBasic with extensions such as, KNNWithMean and KNNBaseline.

I. Ensemble Learning

Model	Weights
Global Mean	-
User Mean	-
Item Mean	-
BaselineOnly	-0.537
SlopeOne	-0.199
MF SGD	-
MF ALS	-
PyFM SGD	0.406
PySpark ALS	0.001
SVD	0.006
SVD++	0.656
KNNBaseline User Based	0.522
KNNBaseline Item Based	0.142

Table I
WEIGHTS OF THE MODELS.

We blend our models by giving weight to each of them and tuning the weights by using ridge regression. This approach increases the performance prominently since different models compensate each other in the process of prediction

of an element. In this blending processs, firstly, we tuned hyperparameters of each model. We used gridsearch method combined with cross validation. Depending on the algorithm, we used 3 or 5 fold cross validation since the computational complexity of certain algorithms are notably high. Because of the same reason, iteration of the algorithms also changed accordingly. After tuning the hyperparameters, we trained our algorithms and optimized the prediction of each model using ridge regression.

Model	Hyperparameters
MF SGD	gamma = 0.008 # of features = 10 lambda (user) = 0.03 lambda (item) = 0.35 # of epoch = 30
MF ALS	# of features = 20 lambda (user) = 0.1 lambda (item) = 0.7 stop criteria = 10^{-4}
PyFM SGD	# of factors = 20 iteration = 200 learning rate = 0.001
PySpark ALS	rank = 20 lambda = 0.1 iteration = 24
SVD	# of epoch = 30 learning rate = 0.01 regression rate = 0.01
SVD++	# of epoch = 30 learning rate = 0.01 regression rate = 0.01
KNNBaseline User Based	k = 300 # of epoch = 30 baseline = pearson
KNNBaseline Item Based	k = 60 # of epoch = 30 baseline = pearson

Table II
HYPERPARAMETERS OF THE MODELS.

IV. RESULTS AND DISCUSSION

Model	RMSE
Global Mean	1.1196
User Mean	1.0288
Item Mean	1.0929
BaselineOnly	1.0016
SlopeOne	1.0041
MF SGD	1.0006
MF ALS	1.0348
PyFM SGD	0.9983
PySpark ALS	1.2654
SVD	1.0113
SVD++	0.9951
KNNBaseline User Based	0.9990
KNNBaseline Item Based	1.0081
Blend	0.9905

Table III
RMSE OF THE MODELS.

We use global mean, user mean and item mean algorithms as our benchmarks. In Table III, we provided the RMSE values of our results. As it is seen, BaselineOnly, SlopeOne and SGD algorithms perform slightly better than the baseline whereas ALS algorithm performs poorly. We obtain best result with SVD++ model from Surprise library. In addition to that, SVD and KNN models also provided successful results. We used different combinations of models in blending process to understand their effect on performance and obtained optimum solution by blending BaselineOnly, SlopeOne, PyFM SGD, PySpark ALS, SVD, SVD++, KNNBaseline User Based, and KNNBaseline Item Based. We observed that using KNNBaseline algorithms with user and item focus have a huge effect on blending performance. Our explanation to this is such that these algorithms are better than each other in different items and users so that blending increases the performance. Our score in *CrowdAI* is 1.018. We notice the increase in the RMSE according to our local RMSE. This behaviour can be explained by the complexity of blending process that introduce an overfitting into the model. According to our research, there are other ensemble learning methods and collaborative filtering algorithms that can be used in this dataset. This could be the future work of this project.

V. SUMMARY

In this project, sparse dataset that contains ratings of $10,000$ users about $1,000$ movies is used to recommend movies to the users by predicting the ratings of movies according to users. User participation, validity and consistency is checked and baseline models are applied to the dataset. Then, more sophisticated collaborative filtering algorithms are applied by using variety of libraries. Finally, algorithms are blended to increase the performance of prediction by applying weights to each model. With this approach, we obtained 1.018 RMSE score in*CrowdAI*.

REFERENCES

[1] H. D. Breese, J. S. and C. Kadie, "Empirical Analysis of Predictive Algorithms for Collaborative Filtering," pp. 43–52, 1998. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1301/1301.7363.pdf

[2] Y. Koren, "The BellKor Solution to the Netflix Grand Prize," 2009, [Online; accessed 20-December-2018]. [Online]. Available: https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf

[3] N. Hug, "Surprise, a Simple Recommender System Library for Python," 2016, [Online; accessed 20-December-2018]. [Online]. Available: http://surpriselib.com/

[4] A. M. Daniel Lemire, "Slope one predictors for online rating-based collaborative filtering," 2007, [Online; accessed 20-December-2018]. [Online]. Available: http://arxiv.org/abs/cs/0702144

[5] CoreyLynch, "pyFM: Factorization Machines in Python," https://github.com/coreylynch/pyFM, 2016.

[6] Y. Koren, "Factor in the neighbors: scalable and accurate collaborative filtering," 2010, [Online; accessed 20-December-2018]. [Online]. Available: http://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/a1-koren.pdf