**Programming for Data Analytics – 2019/2020**

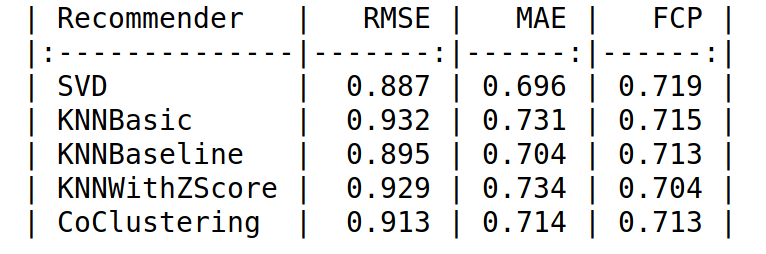
**Project Competition Report**

The goal of this report is to outline the procedures and techniques used to generate my submissions for the project competition of the “Programming for Data Analytics” class of 2019/2020.

In order to generate the predictions required by this project, I have used two different libraries, Surprise and Graphlab, running then some experiments with different algorithms and systems available in the libraries. Given that the experiments based on the same library share most of the code, I have divided this report in two main sections, one per library. Every section outlines first the general procedure, comparing then the results of different experiments.

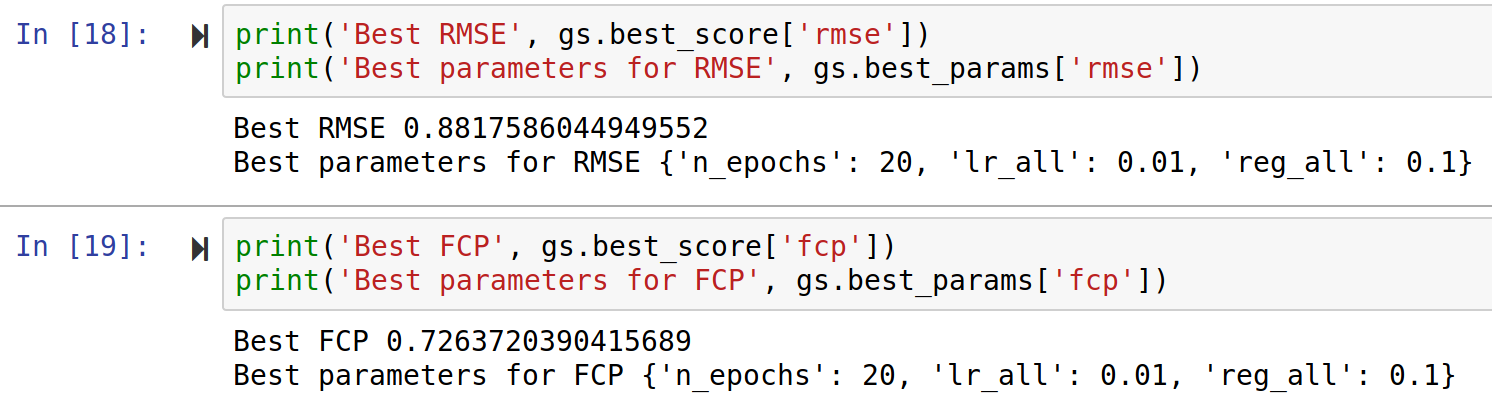
**The Surprise library**

The first experiments I have decided to run were based on the Surprise library, given that it has been previously used in class for different exercises. The two experiments based on this library used the SVD and SVD++ algorithms, since an in-class comparison on the competition’s training data set showed how they were the most suitable ones for our scenario, as shown also by the results summarized in the following table:



In both experiments, the first step is to read the input training and test data sets, operation performed using the Pandas library. From the training data set, the full dense matrix is generating, pivoting the original data set on the userID and itemID columns. This operation is needed to understand which movies have not been seen yet by the different users, so that we can avoid predicting unuseful ratings.

We then extract, from the test data set, the list of users we are interested into for our predictions. Next, we generate the train set, using the reader object, with the rating scale set from 1 to 5, and loading and building the full train set, as shown also in class.

We can then create the recommender system, based on the SVD or SVD++ algorithm, using the generated train set. For the SVD experiment, the most suitable parameters have been found applying the grid search procedure, executed as part of the in-class quiz, with the following results:

Unfortunately, for the SVD++ I was not able to complete the grid search procedure due to some hardware limitations and for this reason I used the default parameters set by the library.

After creating the recommender object, the model is then trained using the generated train set.

Next, we loop through the list of test user and, for each of them, we predict the rating for all movies that he/she has not watched yet. Those movies are represented by a NA in the dense matrix rows, with the movie id corresponding to the column’s name. The predicted ratings are then added in a dictionary having as key the corresponding movie id. After predicting all ratings for the single user, we sort the dictionary based on the value (hence the rating), taking then the first 10 movies, therefore the 10 highest rated movies. We generate then the formatted string and we add it in the test data frame, in the cell corresponding to the analysed user.

At the end, we obtain a data frame containing, for each user, a string representing the IDs of the top 10 recommended movies. At this point, we can use Pandas to simply export the data frame to a CSV file with the desired specification, that is then ready to be submitted to Kaggle for the evaluation.

In both cases, when using SVD or SVD++, I was able to obtain results over 0.02 Mae score, with the SVD++ scores being slightly higher. In this set of experiments I did not use the information about the movies available in the “content” data set, since I was unsure about how to use them to increase the precision when using these algorithms. The files corresponding to the two experiments performed using surprise are available in the GitHub repository (please see reference).

**The Graphlab library**

The second library that has been used in this project is the Graphlab library. This library is capable to create an effective recommender system without writing large portions of code. I run different experiments, starting with the standard recommended system built by the system and then trying to enforce some specific algorithms.

In all cases, the first step is clearly to read and parse the input file. In this case, the training and content data set are read using the parse\_csv function offered by Graphlab, since it returns a specific object that is then required by the recommender system function. On the other hand, the test data set is still read using Pandas, always to ensure a simple manipulation of the data frame.

After reading the input files, we extract from the test data frame the IDs of the users we are interested into. We then create the recommender system using the chosen function (standard or enforcing a type, as outlined below), training it with the input data set and specifying the user and item column names and the target of the prediction (in our case, the rating column).

In the case of Graphlab, we can then simply iterate over the list of the test users, calling the recommend functions and passing the corresponding user’s ID. The library is automatically able to determine the items that have not been ranked yet and to predict the corresponding value, returning then the top 10 items, exactly the one we are interested into. Following the same idea that was used for the surprise related experiments, we create the formatted string from the item IDs returned by the recommender system, adding them in the test data frame, using Pandas.

We then export our results to a CSV file, always using Pandas.

In the case of Graphlab, I started using the standard recommender system that is built by the library, without adding information related to the movies (and available in the content file) or specifying the algorithm. It turned out that this function tries to determine automatically the best algorithm to apply, analysing the training data and choosing between the factorization recommender and the item similarity one. Since the beginning, results obtained with this recommender have been consistently higher than the one obtained using SVD or SVD++, being always over 0.05.

After running the standard recommender, I tried to enforce manually the factorization recommender since, at that point, I hadn’t yet discovered the automatic mechanism of the standard function. For this reason, I got similar results and, of course, no improvements.

The last experiment I run consisted in adding some extra information to the standard function, using so the information available in the contend data set: the information were read using the same function adopted for the training data set and they were passed as a parameter to the standard recommender function. However, this change had not a relevant impact on the final Mae score, since I obtained scores similar to the previous experiments.

**Notes**

At the beginning, I tried to change the algorithm multiple times, when trying to use SVD, given that an error in the use of the sorting function did not enforce the correct order in the final dictionary. Those changes has been made only for debugging purposes, since the first idea was to use SVD, and they appear in the history of my submissions. However, the change was only in the algorithm and was a temporary change made overwriting the SVD file, until I finally spotted the error.

**GitHub Reference**

All source related to this project and experiments can be found as commented Jupyter Notebooks in the PDA19 repository, available at the following address: https://github.com/sbettid/PDA19

Surprise related experiments:

* PDA19 Challenge – SVD predictions.ipynb
* PDA19 Challenge – SVD++ predictions.ipynb

Graphlab related experiments:

* PDA19 Challenge – Graphlab predictions.ipynb
* PDA19 Challenge – Graphlab\_factorized predictions.ipynb
* PDA19 Challenge – Graphlab\_itemInfo predictions.ipynb

The data folder contains the input data sets provided in order to complete the competition, while the generated folder contains the exported results CSV files.