Recommender system for MovieLens dataset

by Vladislav Urzhumov
(v.urzhumov@innopolis.university)

Part 0: Introduction

Present report aims to describe and evaluate the solution of recommendation systems problem of predicting some movies that the user has never watched before but most probably will like to. The whole task is built on the rating system: user with *user_id* can assign a rating to a movie he (most probably) watched, and rating is one of the numbers from the set [1, 2, 3, 4, 5], with 5 being the best rating (excellent), and 1 being the worst (poor). The data analysis and solution building are described in the following parts of the report.

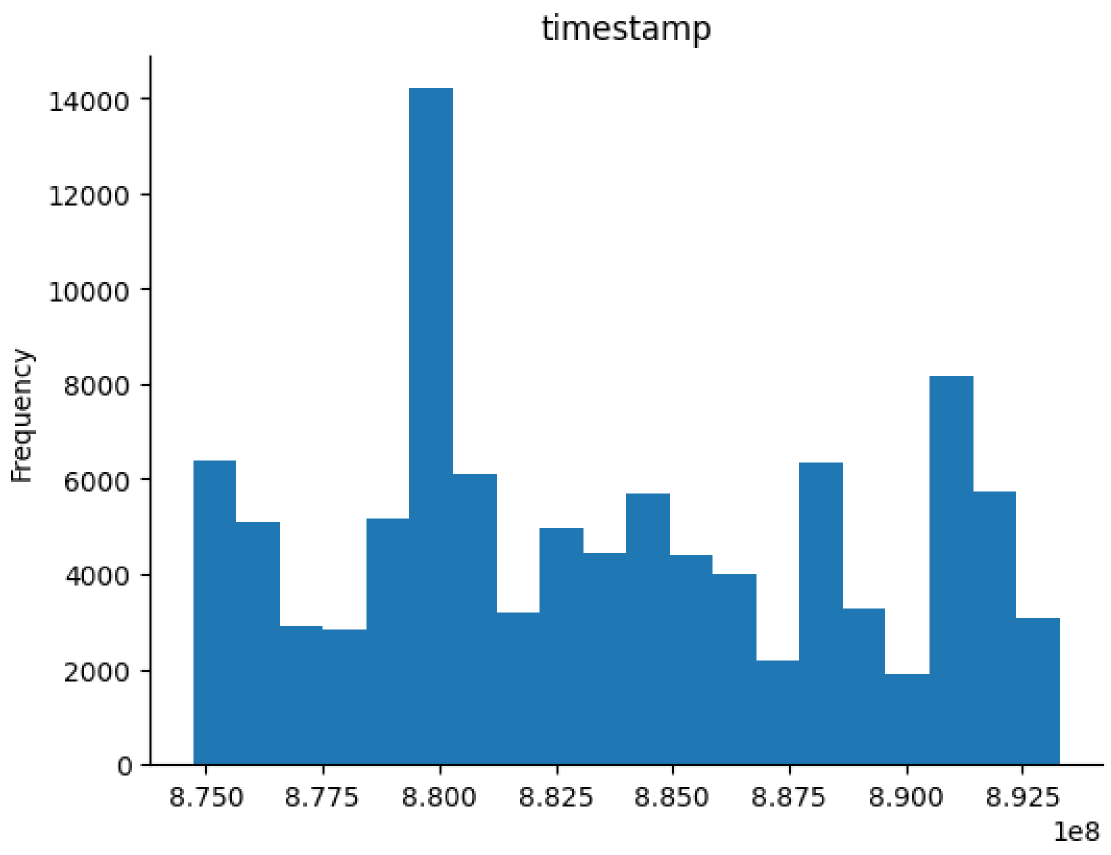
Part 1: Data analysis

Movie lens dataset consists of several files, as described in the task description (data/external/task/README.md).

The main file with whole data of user ratings for films is “u.data”, tab-separated-value formatted. We can load it to pandas.DataFrame directly.

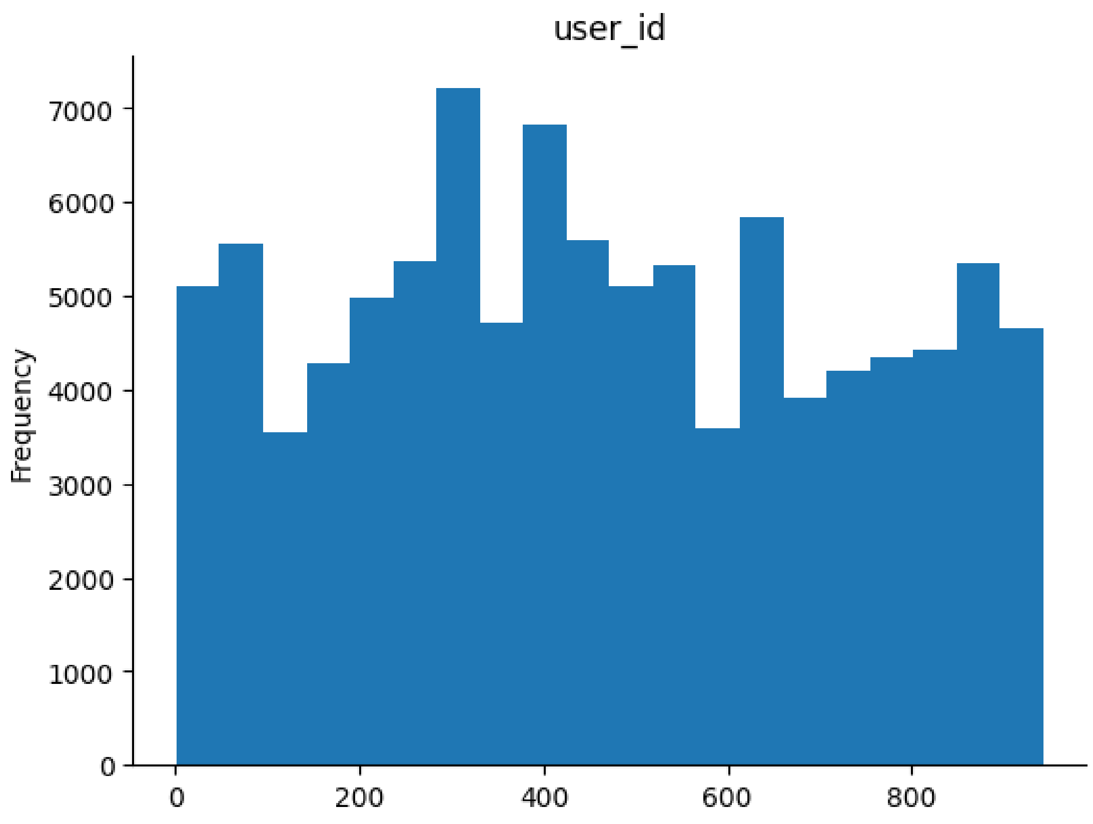
The data is [user_id | movie_id | rating | timestamp] table.

Timestamp stands for time of user’s action: at that specific time *user_id* user has given *rating* to *movie_id* movie. We may need it later to sort data by time, e.g. to predict latest user’s ratings. The distribution of timestamps:

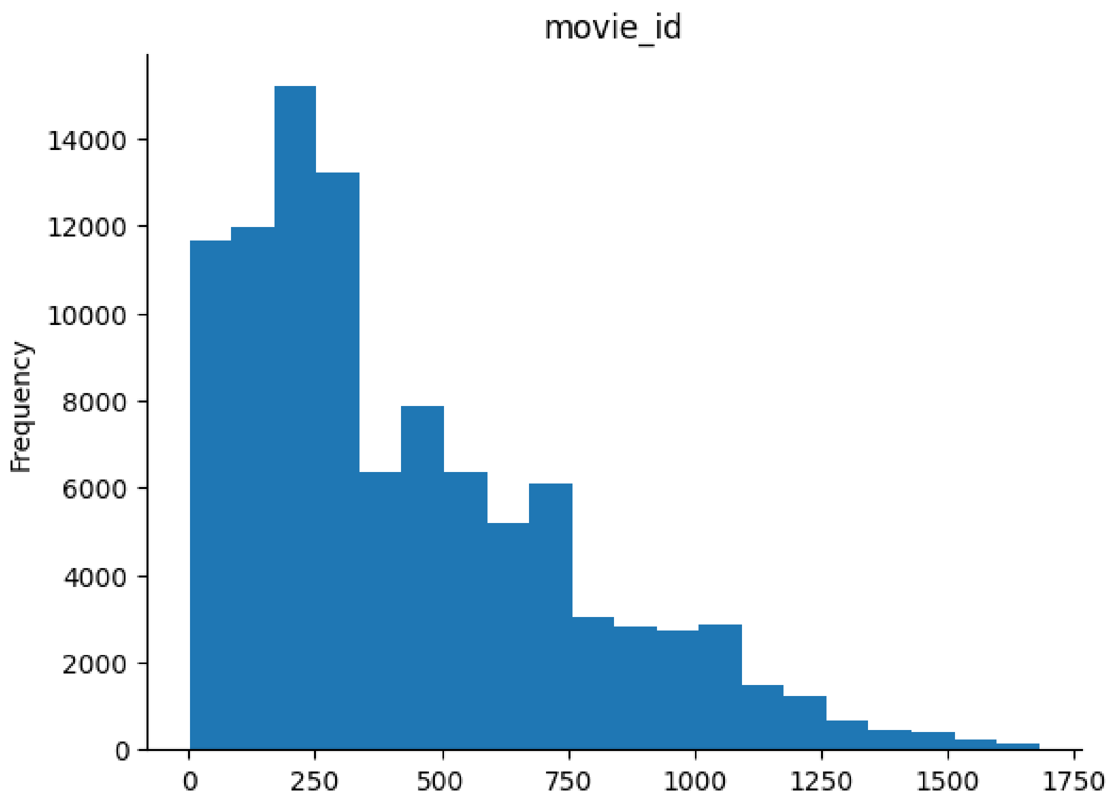


Movie_id and user_id columns are essential, they are indices of certain movie and certain user, correspondingly. Metadata can be found in additional files of MovieLens. However, let us consider distributions.

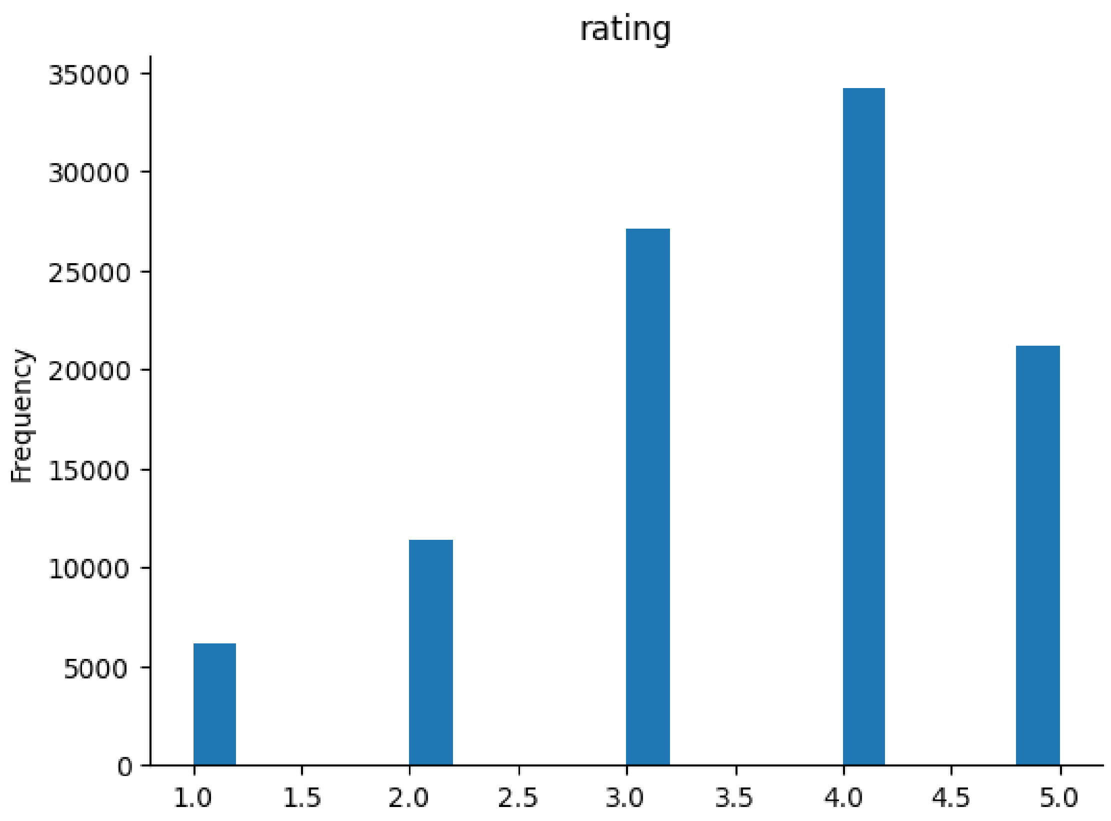
User activity (number of ratings that user put to the system):



Next, movie popularity (more ratings considered more popular):



Finally, ratings distribution:



Ratings 3 and 4 are most common. People love to rate films as “average” or “positively average” (i.e. “quite good, not perfect”). Probably, our models will be more biased to put ratings between 3.0 and 4.0.

Exploration of train-test subsets of the dataset (u1.base and u1.test) is present in the 0.1 explorative Jupyter notebook.

To work with data, we will need to process it in the format of sparse matrix, with user_id and movie_id as rows and columns, and ratings as value. Timestamp is used for time-dependent train/test split.

Part 2: Note on solution building and copyrights

Project solution is built for Practical ML&DL course at Innopolis University as an Assignment 2 solution, with the rules described in (data/external/task/README.md). At this point, I need to note that partially some blocks of code may be inspired by or copied from open-source projects and/or articles. Links to those sources will be provided both in code, notebooks and present report. Authors of these sources either mentioned or provided a license allowing to copy and/or use code for non-commercial, study-only or personal purposes. The code development approach is solely under the rules of given assignment description, since all of the sources are provided. All the data exploration, notebooks, reports, code files, descriptions, docstrings, figures, rules, documentations, and whole solution idea (including architecture design, inter-file logic, etc.) are made by the one and only author of this project (stated right below the report main title) for Innopolis University. One can contact the author of current project via e-mail address stated below the main title or in person in case of any questions.

Part 3: Evaluation

To address the performance measurement problem, one can utilize either rating prediction accuracy measurement (rating is target value, metric is regression metric) or ranking quality metrics for ranking systems, such as ndcg and recall. Since overall rating prediction ability is crucial for this task, I prefer to stick to rating measurement metrics. To avoid chasing lone metric value and reduce sensitivity to outliers, I have utilized two metrics: Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). The mainly considered metric is RMSE in the current project, thus we will mostly focus on it, and then secondly on MAE.

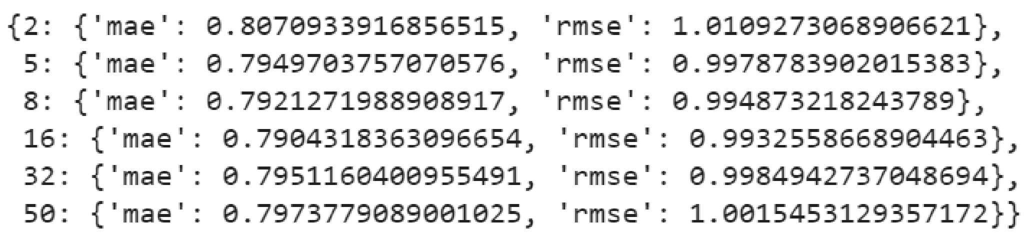
Part 4: Baseline

MovieLens is a popular dataset to start working with recommender systems. Small research shows the vast amount of solutions existing for this problem, starting from simple averaging of user ratings to Machine Learning models. The easiest solution However, we want to create a relatively strong baseline solution to beat later with a more advanced model.

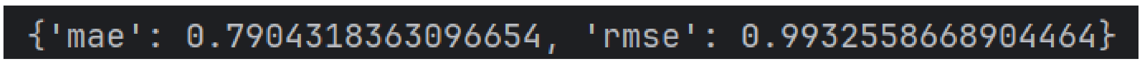
As a baseline we will consider SVD approach. Code for the baseline is present both in main body of the project and Jupyter notebook 1.0.

One can test the performance of the baseline solution (results are generated quickly, in 1-2 minutes for find_best_k function to iterate over proposed n_components of SVD). Also, recommendation method is present, it takes user_id and k (number of films to recommend) as parameters and produces top k suggestions with movie_id and predicted rating that most probably user will give to that movie.

Metrics are both calculated in notebook and accessible via code run with baseline argument set to True. Here are calculated metrics:



```
{2: {'mae': 0.8070933916856515, 'rmse': 1.0109273068906621},
 5: {'mae': 0.7949703757070576, 'rmse': 0.9978783902015383},
 8: {'mae': 0.7921271988908917, 'rmse': 0.994873218243789},
 16: {'mae': 0.7904318363096654, 'rmse': 0.9932558668904463},
 32: {'mae': 0.7951160400955491, 'rmse': 0.9984942737048694},
 50: {'mae': 0.797377908901025, 'rmse': 1.0015453129357172}}
```



```
{'mae': 0.7904318363096654, 'rmse': 0.9932558668904464}
```

(first dictionary has n_components as keys and is a result of so-to-say grid search of that hyperparameter).

Part 5: Advanced technique

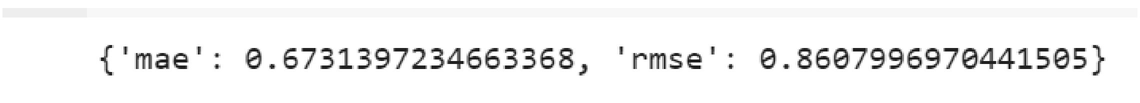
Research on the recommender systems topic has led me to a paper on Global-Local Kernel model utilizing AutoEncoder (AE) (with the local kernelized weight matrix) and Global Kernel-based matrix, solving the problem in two steps: firstly, Pre-training the AE, and secondly, Fine-tuning the Global Kernel. Authors state that the performance of the developed approach is exceptional, outplaying most State-of-the-Art models by RMSE metric. Since we are focused on RMSE (as was mentioned in Part 3 of present report), I have used the official implementation of the GLocal-K model (and it's pytorch library adapted version) to beat the baseline and obtain better solution. However, I have modified it to meet the assignment needs

Moreover, an unofficial survey among Innopolis University students taking the aforementioned course and solving the aforementioned assignment showed, that the performance of this modified GLocal-K is truly exceptional, outplaying most of the models others used to solve that task (even those which considered metadata), in terms of RMSE metric.

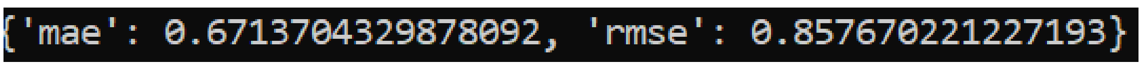
Worth mentioning that none of the metadata about users or movies was used, as a challenge to reach high scores on the sparse rating matrices only. Real-world recommender systems field problems have little to no relevant metadata to work with, thus the focus should be made on the pure ratings. Possible enhancements considering metadata may be added on top of the GLocal-K model in the further development steps to obtain hybrid model.

The screenshots below show the best scores reached by the GLocal-K model on u1.test. Thus, they can be compared to the baseline performance metrics in a straightforward way.

Screenshot with white background shows the performance reached by the model trained in Jupyter notebook; black-backgrounded screenshot shows the performance of the model trained locally on the desktop. Note that minor differences in results with the same hyperparameters used are a subject to depend on random weights instantiation.



```
{'mae': 0.6731397234663368, 'rmse': 0.8607996970441505}
```



```
{'mae': 0.6713704329878092, 'rmse': 0.857670221227193}
```

Part 6: Model advantages and disadvantages

Baseline (SVD) approach advantages include small inference time, relatively good evaluation results (compared to the simplest methods found during research, such as averaging user's ratings, etc.), ease of implementation and mathematical simplicity (relatively). Disadvantages are simply low improvement capability (can not do much better using the same data, since no training)

However, Global-Local Kernel model outperforms most of the existing solutions in terms of metrics we used, and it's advantages include exceptional performance, trainability, scalability and improvement capability (hyperparameters and ML-specific techniques may increase performance even more). Disadvantages include difficulties of implementation, low example availability in the field, and relatively large training time (compared to non-trainable approaches).

Both approaches use no metadata, this can be treated as advantage (no additional data needed, plus improvement capability in terms of stacking another model on top to consider the metadata and give more accurate predictions)

Part 7: Training process

Training of the GLocal-K model was made both locally and with cloud services, such as Google Colab. Both training methods showed similar results and took around 30 minutes for 110 epochs.

The obtained weights of the best model are uploaded on the hugging face and can be accessed via link. To use them, put weights uploaded by the latest commit to the data/interim/experiment/checkpoints directory all together. Do not forget to set the project_path variable in the models/utils/config.py to represent the path to the root of your project.

No training is required for SVD, just provide train and test sets to evaluate thee approach

Part 8: Results

I implemented, trained and evaluated two different approaches to solve present recommendation systems problem on MovieLens-100k dataset, and obtained promising results. Baseline was strong, however it was outperformed by GLocal-K model made with the use of original implementation provided by authors. The table below shows comparisons between different versions of two approaches, in terms of utilized metrics.

Approach	**Version**	**MAE**	**RMSE**
SVD	N_components=50	0.7974	1.002
SVD	N_components=16	0.7904	0.9933
GLocal-K	10+20 epochs training	0.7735	0.9741
GLocal-K	30+80 epochs training	0.6714	0.8577

The journey was interesting and provided a bunch of valuable experience. Huge gratitude to the Practical Machine Learning & Deep Learning course instructors for this beautiful assignment and for the whole course!