Collaborative Filtering

by Juan Manuel Alonso

Installation

```
$  pip3 install -r requirements.txt
```

Running

Custom hyperparameters in a textfile i.e. *"./configs/config.txt"*.

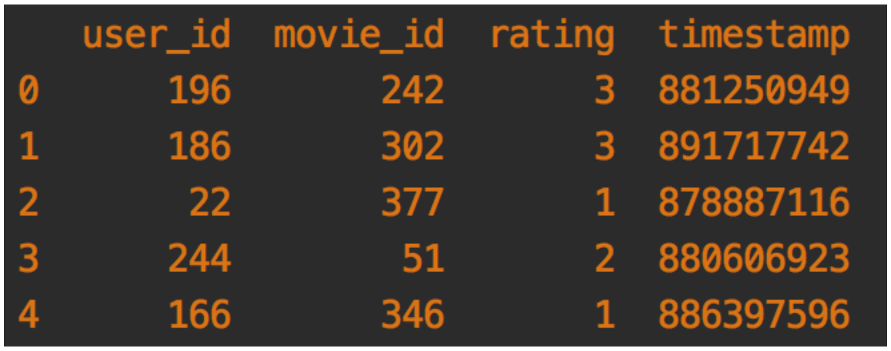
```
$  python3 experiments.py ./configs/config.txt
```

A *results* folder will contain a timestamp directory with the latest results, depending on the dataset.

Datasets

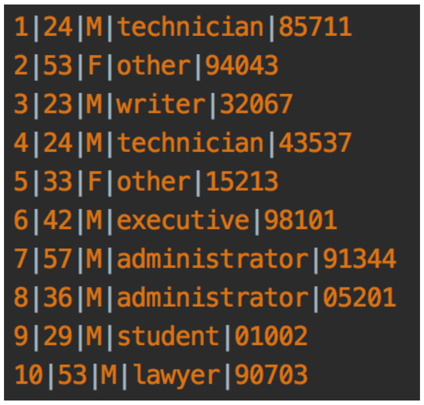
- MovieLens (with 100k and 1m ratings) (https://grouplens.org/datasets/movielens)

The size of the data frame for the small 100K ratings example is (100000, 4). The first rows of this dataset's dataframe are shown below:



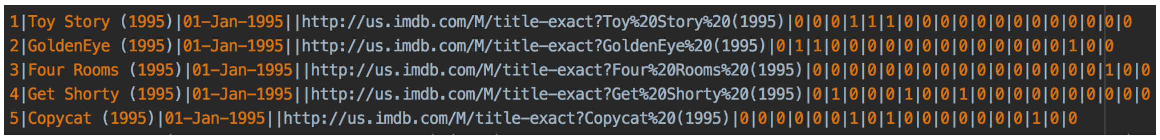
```
   user_id  movie_id  rating  timestamp
0      196       242       3  881250949
1      186       302       3  891717742
2       22       377       1  878887116
3      244        51       2  880606923
4      166       346       1  886397596
```

This dataset is often used as a benchmark for collaborative filtering experiments. It is composed of 4 features of ratings done by a user: user_id, movie_id, the 5-star rating given and the timestamp. There are 943 users, 1682 items (or movies) and 100000 ratings in it. Among others, the genres present in the dataset are Action, Comedy, Crime and Drama. A sample of the users that rated such movies is given below:



```
1|24|M|technician|85711
2|53|F|other|94043
3|23|M|writer|32067
4|24|M|technician|43537
5|33|F|other|15213
6|42|M|executive|98101
7|57|M|administrator|91344
8|36|M|administrator|05201
9|29|M|student|01002
10|53|M|lawyer|90703
```

The same goes for the movies, as a snippet of them is included:



```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0
2|GoldenEye (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?GoldenEye%20(1995)|0|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0
3|Four Rooms (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Four%20Rooms%20(1995)|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0
4|Get Shorty (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Get%20Shorty%20(1995)|0|1|0|0|0|1|0|0|1|0|0|0|0|0|0|0|0|0|0
5|Copycat (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Copycat%20(1995)|0|0|0|0|0|0|1|0|1|0|0|0|0|0|0|0|1|0|0
```

Following is a histogram of the ratings present in the dataset, from which one can observe their distribution is not uniform.

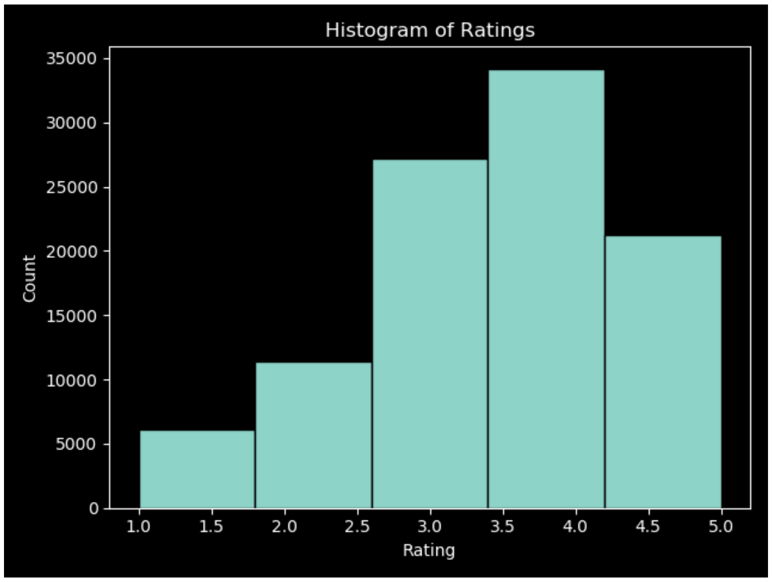


FIG. 1: HISTOGRAM OF RATINGS, 5-STAR, INTEGER ONLY. MOVIE LENS 100K DATASET

Techniques

- User-based kNN (k=40)
- Item-based kNN (k=40)
- Autoencoder (Neural Network Model-Based Approach)

Results

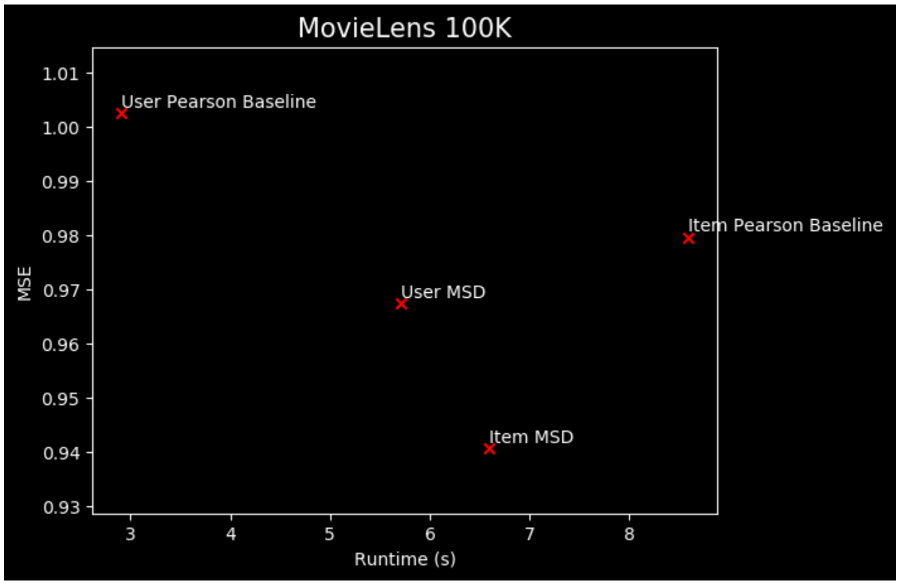


FIG. 2: RUNTIME V MSE ERROR FOR DIFFERENT ALGORITHMS. MOVIE LENS 100K DATASET

The comparison of the performance between the different algorithm is done in terms of the effectiveness of the recommendations and in the efficiency (i.e. runtime).

From Fig. 2, once can notice that the performance of the different algorithms on a determined test set is not homogenous. This test set derives from a 80-20 split (training-testing) of the dataset.

All algorithms, wether user or item based, were implemented by the scikit-surprise library[1], using kNN algorithm internally, with k=40, and a similarity measure is defined for each one. In this experiment, the similarity measures selected were MSD (Mean Squared Difference similarity between all pairs of users or items), and Pearson Baseline (the shrunk Pearson correlation coefficient between all pairs of users or items using baselines for centering instead of means[2]).

[1] http://surpriselib.com

[2] https://surprise.readthedocs.io/en/stable/similarities.html#module-surprise.similarities

The user-based algorithms perform better in terms of runtime for the utilized test set, but the best-performing algorithm from Fig 2 in terms of mean squared error is the item-based kNN MSD approach.

Another model-based algorithm was used for the purpose of finding better performance. In this way, a shallow neural net model was used, derived from a blog[3], which in turn was inspired by a research paper[4].

This is a machine learning method, also known as an *autoencoder*, is fed with all the movie ratings for each user (with 0 if the user did not rate it yet), and in this fashion, expect some generalization power at the output layer for ratings of movies that user has not rated.

The *autoencoder* approach was tweaked to have 1682 input and output nodes, the exact number of movies. Its batch size was increased to 20. For this dataset and its larger version, a learning rate of 0.001 was used, together with 32 hidden neurons, which showed to have the best performance.

With a test time of 0.0038 seconds and a MSE of 0.626, this autoencoder approach, which was not included in the figures so as to better distinguish the differences of the previous methods, is clearly the best performing technique, with the training phase duration as its tradeoff.

[3] https://medium.com/@connectwithghosh/recommender-system-on-the-movielens-using-an-autoencoder-using-tensorflow-in-python-f13d3e8d600d

[4] http://users.cecs.anu.edu.au/~akmenon/papers/autorec/autorec-paper.pdf

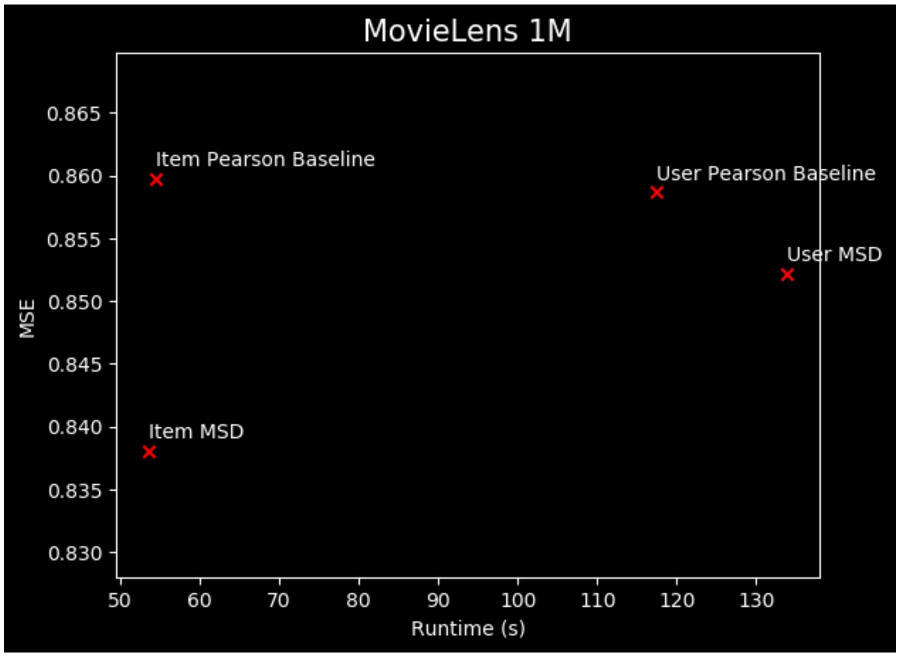


FIG. 3: RUNTIME V MSE ERROR. MOVIE LENS 1M DATASET

The same procedures were done with the larger version of the MovieLens dataset of 1 million ratings. From Fig. 3 one may notice that the item-based algorithms clearly perform better in terms of runtime, with the best-performing algorithm in terms of the mean squared error is once again the item-based kNN MSD approach.

It may be said from Fig.3 that the user-based approaches do not scale as well for much large data, since they were more efficient in the smaller dataset.

The *autoencoder* approach was tweaked to have 3706 input and output nodes, the exact number of movies. Its batch size was increased to 80. It had 13 times the runtime compared to its smaller run, and a slightly better MSE of 0.583. The result was again not included in the figures so as to better distinguish the differences of the previous methods.