|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | | | | |
|  | |  | | | |
| Project Topic: | | Implementation of Recommender System using KNN and Matrix Factorization | | | |
| Module: | | Report | | | |
| Deliverable:  Course: | | Programming Assignment 2  CSE 6363-004 - Fall 2020 | | | |
| Version: | | | [1.0] | Date: | [10/18/2020] |

Prepared by:

Rahaman, Fahad Ur (Student ID: 1001753107)

TABLE OF CONTENTS

[1. KNN IMPLEMENTAtION AS A RECOMMENDER SYSTEM 2](#_Toc492333746)

1.[1 APPROACH USED FOR KNN AS A RECOMMENDER SYSTEM 2](#_Toc492333748)

[1.2 MATRIX FACTORIZATION TO THE DATASET 6](#_Toc492333749)

[1.3 MAXIMUM DATASET THE RECOMMENDATION SYSTEM CAN USE 7](#_Toc492333749)

[1.4 TIME COMPLEXITY OF RECOMMENDATION SYSTEM 8](#_Toc492333749)

[1.5 PERFORMANCE OF THE RECOMMENDATION SYSTEM 9](#_Toc492333749)

[1.6 SCALING UP OF RECOMMENDATION SYSTEM 10](#_Toc492333749)

[1.7 SCALING UP OF RECOMMENDATION SYSTEM 10](#_Toc492333749)

[REFERENCES 11](#_Toc492333762)

# 1. KNN IMPLEMENTATION AS A RECOMMENDER SYSTEM

## 1.1 Approach Used For KNN As A Recommender System

In Order to Implement KNN Algorithm as a Recommender System, we must follow the below 3 STEPS:

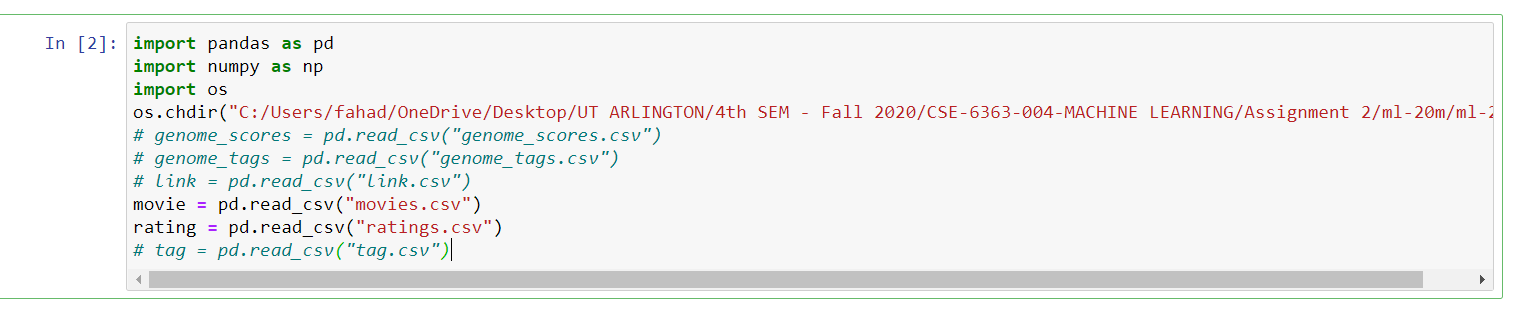
* We Need to find the distance between 2 users
* From one point or one user, we need to find the distance to all other users
* Based on the K-Nearest Neighbors we need to recommend movies to similar users

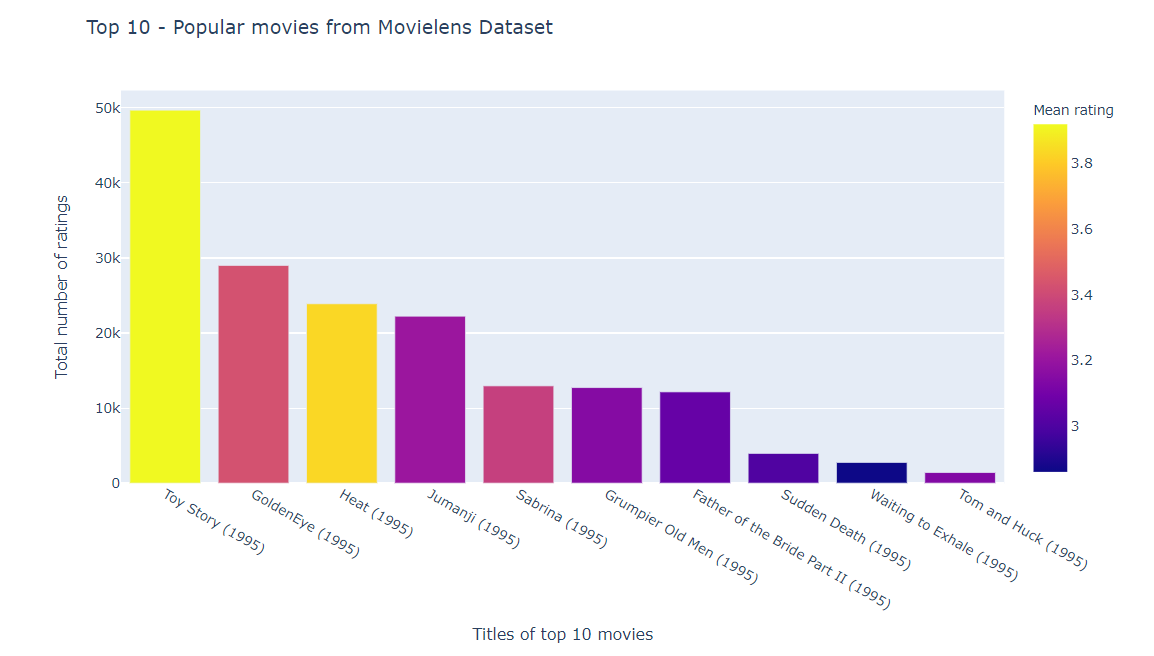
In Our Implementation of K-Nearest Neighbors, we have followed the below steps:

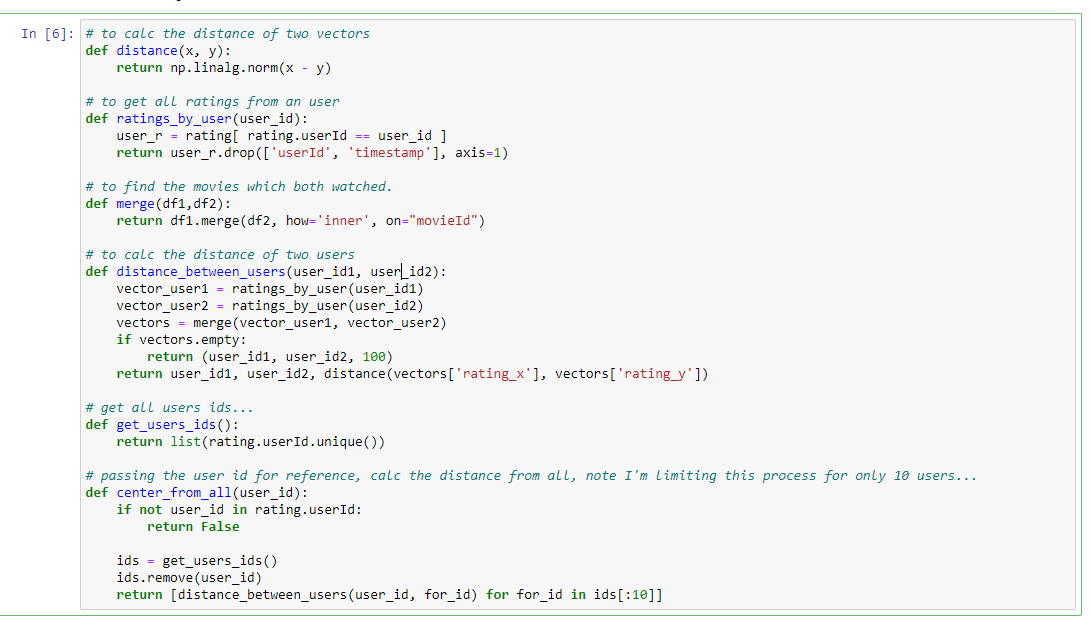
* Firstly, we load the dataset (“movies.csv” and “ratings.csv”) from movie lens dataset using “read\_csv”
* Then we calculate the average and total ratings for each movie id
* Calculated the distance of two vectors, got all ratings from an user
* Found the movies which 2 users watched
* Calculated the distance of two users
* Made recommendation based on similarity between the users
* Finally, we will evaluate the algorithm using map@k (Mean Average Precision) metrics to calculate the performance of the recommendation system

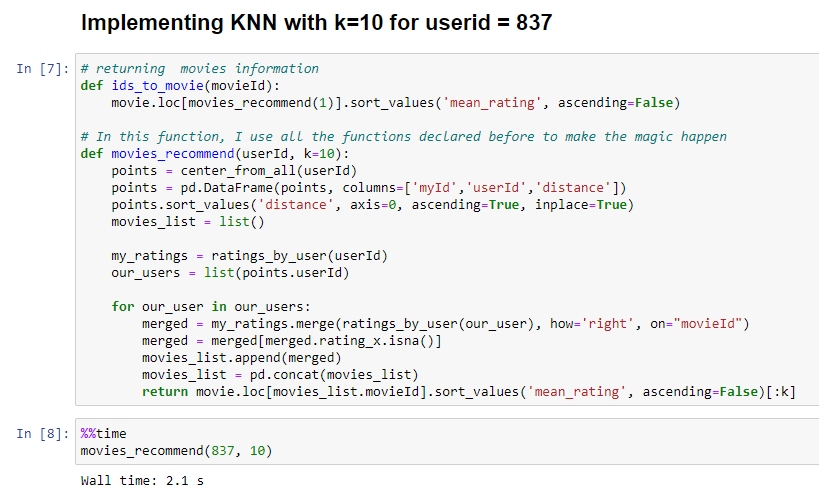
Below are the Screenshots of code implemented in python

Please find the complete implementation of the python code in “Programming Assignment 2 - CSE 6363-004\_FAHAD UR RAHAMAN”











## 1.2 Matrix Factorization to the Dataset

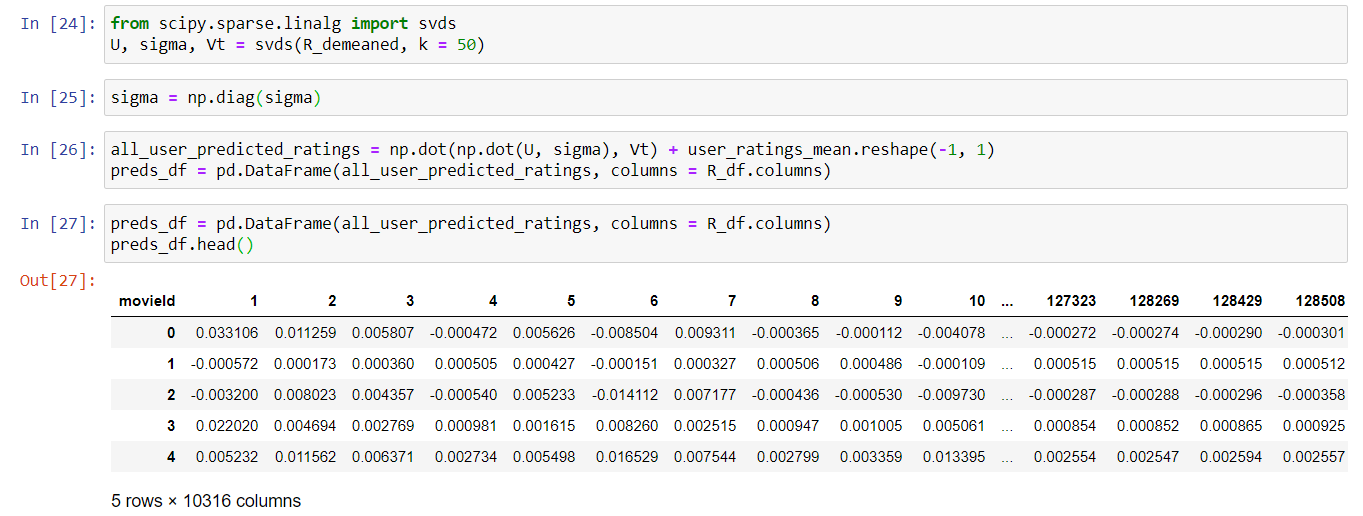
Matrix factorization is the breaking down of one matrix in a product of multiple matrices. So, what is singular value decomposition (SVD)? At a high level, SVD is an algorithm that decomposes a matrix R into the best lower rank (i.e. smaller/simpler) approximation of the original matrix R. Mathematically, it decomposes R into a two unitary matrices and a diagonal matrix:

R = U Σ V^{T}

where R is user’s ratings matrix, U is the user "features" matrix, Σ is the diagonal matrix of singular values (essentially weights), and V^{T} is the movie "features" matrix. U and V^{T} are orthogonal and represent different things. U represents how much users "like" each feature and V^{T} represents how relevant each feature is to each movie.

To get the lower rank approximation, we take these matrices and keep only the top k features, which we think of as the underlying tastes and preferences vectors.

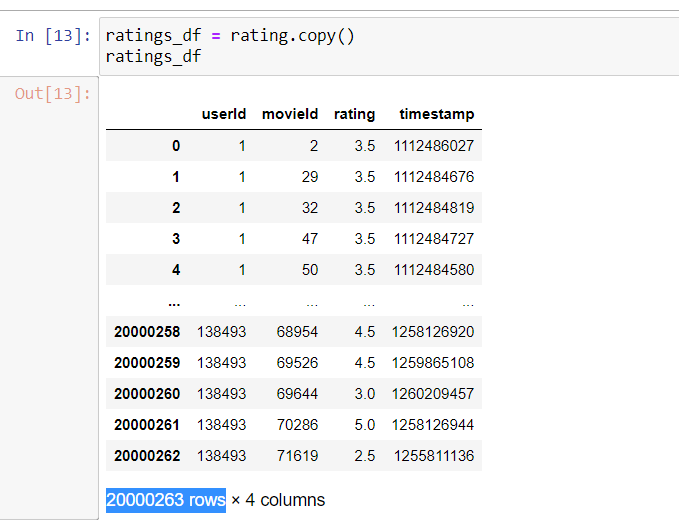
Below is the screenshot of the code where the Matrix Factorization implemented:

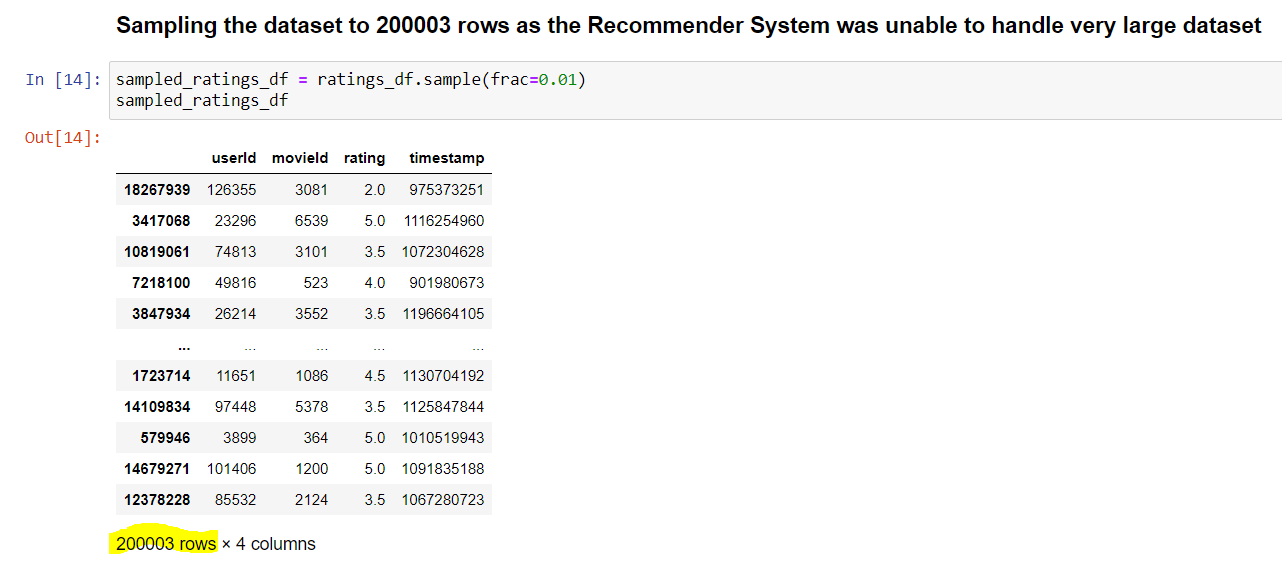


## 1.3 Maximum Dataset the Recommendation System Can Use

Our Recommendation System can only handle dataset up to “**200003 rows**”. Since the Original dataset was containing 20000263 rows, we had to do random sampling to create the subset from the original dataset and provide as an input to our recommendation system.

Please find the below screenshots of the code implemented:



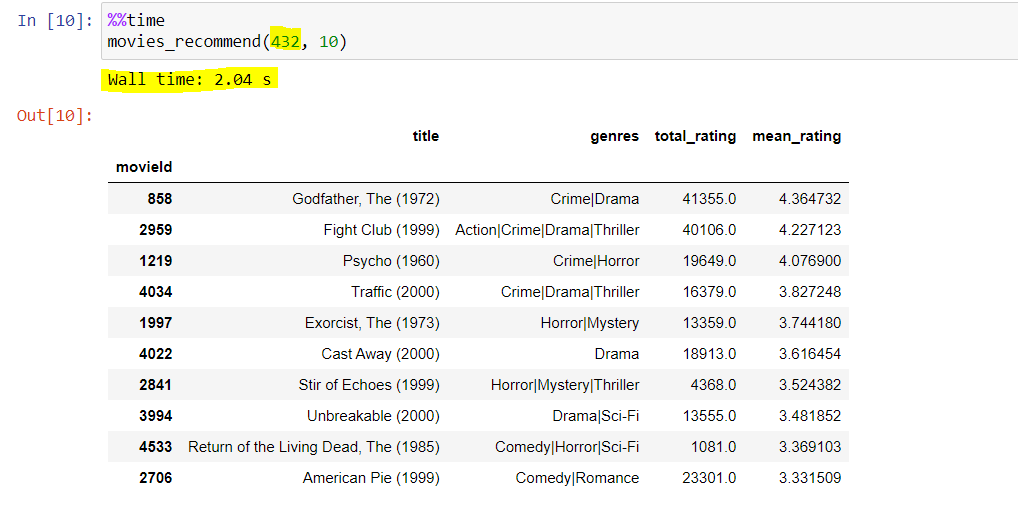


## 1.4 Time Complexity of the Recommendation System

On an average our Recommendation System took around “**2 seconds”** to make 10 predictions for a particular user (Or “userId”).

Please find the below screenshots for the implementation of the code:







## 1.5 Performance of the Recommendation System

In order to evaluate the performance of our recommendation system we calculated the MAP@K(Mean Average Precision)

Below is the formula used to calculate “Average Precision” (AP) and “Mean Average precision” (MAP):

If we are asked to recommend N items, the number of relevant items in the full space of items is m, then:

**AP@N = 1m∑k = 1N(P(k) if kth item was relevant) = 1m∑k=1NP(k)⋅rel(k)**,

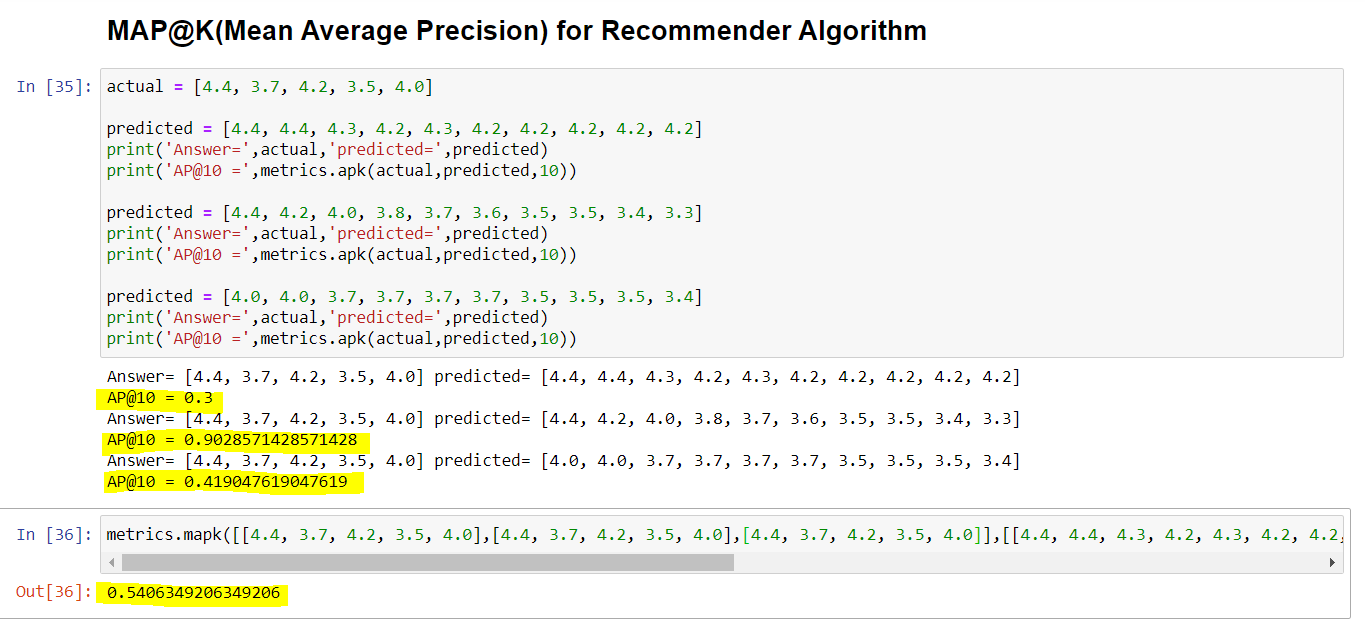
where rel(k) is just an indicator that says whether that kth item was relevant (rel(k)=1) or not (rel(k)=0). I'd like to point out that instead of recommending N items would have recommended, say, 2N, but the AP@N metric says we only care about the average precision up to the Nth item.

MAP@N? It is the average AP@N metric over all |U| users. Yes, an average of an average.

**MAP@N=1|U|∑u=1|U|(AP@N)u=1|U|∑u=1|U|1m∑k=1NPu(k)⋅relu(k).**

After applying this metric, the performance achieved by our recommendation system = **0.5406349206349206**

Below is the screenshot of the code implemented for the evaluation of metric:



## 1.6 Scaling Up of Recommendation System

At large scale, the dataset used for training will no longer fit in memory, even when stored sparsely.

One way to work around this problem is to do training out-of-core, which is when batches of training data are incrementally fed to the model. Surprise and most other recommender system frameworks lack incremental training algorithms, so we have to hand-roll one ourselves.

[SciKit-Learn](http://scikit-learn.org/stable/index.html) has a module called [IncrementalPCA](http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.IncrementalPCA.html). Unfortunately, this model has two problems. First, it does not accept sparse data, so we will have to incrementally feed it dense data. Doing so is ludicrously slow, especially with this number of features and users. Second, the model doesn’t make predictions that are anywhere near as accurate as Surprise or SciKit-Learn’s [TruncatedSVD](http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html). This inaccuracy is probably due to the data sparsity rather than an inherent problem with IncrementalPCA.

The better solution is to use Apache Spark and its built-in MLLib toolkit.

## 1.7 Conclusion

We have seen that we can make good recommendations with raw data based collaborative filtering methods (neighborhood models) and latent features from low-rank matrix factorization methods (factorization models).

Low-dimensional matrix recommenders try to capture the underlying features driving the raw data (which we understand as tastes and preferences). From a theoretical perspective, if we want to make recommendations based on people's tastes, this seems like the better approach. This technique also scales significantly better to larger datasets.

However, we still likely lose some meaningful signals by using a lower-rank matrix. And though these factorization-based techniques work extremely well, there's research being done on new methods. These efforts have resulted in various types probabilistic matrix factorization (which works and scales even better) and many other approaches.

One particularly cool and effective strategy is to combine factorization and neighborhood methods into one [framework](http://www.cs.rochester.edu/twiki/pub/Main/HarpSeminar/Factorization_Meets_the_Neighborhood-_a_Multifaceted_Collaborative_Filtering_Model.pdf).

# References

[1] https://heartbeat.fritz.ai/recommender-systems-with-python-part-ii-collaborative-filtering-k-nearest-neighbors-algorithm-c8dcd5fd89b2

[2] https://www.geeksforgeeks.org/python-pandas-dataframe-sample/

[3] https://beckernick.github.io/matrix-factorization-recommender/

[4] https://www.kaggle.com/mcarujo/knn-movie-recommendation-movielens

[5] https://github.com/beckernick/matrix\_factorization\_recommenders/blob/master/matrix\_factorization\_recommender.ipynb

[6] https://towardsdatascience.com/evaluation-metrics-for-recommender-systems-df56c6611093

[7] https://stackoverflow.com/questions/17197492/is-there-a-library-function-for-root-mean-square-error-rmse-in-python

[8] https://pandas.pydata.org/pandas-docs/stable/user\_guide/merging.html

[9] https://www.kaggle.com/wendykan/map-k-demo

[10] http://sdsawtelle.github.io/blog/output/mean-average-precision-MAP-for-recommender-systems.html

[11] hhttps://pdfs.semanticscholar.org/7d60/90ef21f15f9f1210b6f96664e6a3a0e6b507.pdf

[12] http://datastronomy.com/building-your-recommender-system-at-the-right-scale/