



Comparative Study of Different Collaborative Filter based Recommendation Engines

Submitted by:

Riddhiman Kumar (1729052)

Debasrito Lahiri (1729122)

Hritam Jaiswal (1729130)

Satvik Tiwari (1729155)

Under Guidance from:

Prof. Rajdeep Chatterjee

Contents:

- Introduction
- What is a recommendation engine?
- Scope and objectives
- Types of recommendation engines
- How collaborative filters work
- Memory Based
- Model Based
- Project breakdown
- Implementation
- Observations
- Conclusion

Introduction

- Recommendation Engines have now been in use since quite some time for predicting content for users.
- Common uses include YouTube, Amazon, Netflix etc.
- Based on past user browsing history and reviews of users, new content is recommended to the user based on their personal taste and overall review of the contents.
- It is of various types, the most common being:
 1. *Content based filtering*
 2. *Collaborative filter based*
- Here, we will compare different techniques of collaborative filtering based recommendation engines.
- We will construct the different models and train them on the dataset. Then we will try to predict the user ratings for the remaining data in the dataset.

What is a recommendation engine?

- A recommendation engine is a system that suggests products, services, information to users based on analysis of data.
- The recommendation can derive from a variety of factors such as the history of the user and the behavior of similar users.
- A recommendation engine can significantly boost revenues, Click-Through Rates (CTRs), conversions, and other essential metrics. It can have positive effects on the user experience, thus translating to higher customer satisfaction and retention.
- Let's take Netflix as an example. Instead of having to browse through thousands of titles, Netflix presents us with a much narrower selection of items that we are likely to enjoy. This capability saves us time and delivers a better user experience. With this function, Netflix achieved lower cancellation rates, saving the company around a billion dollars a year.
- Although recommender systems have been used for almost 20 years by companies like Amazon, it has been proliferated to other industries such as finance and travel during the last few years.

Scope and objectives

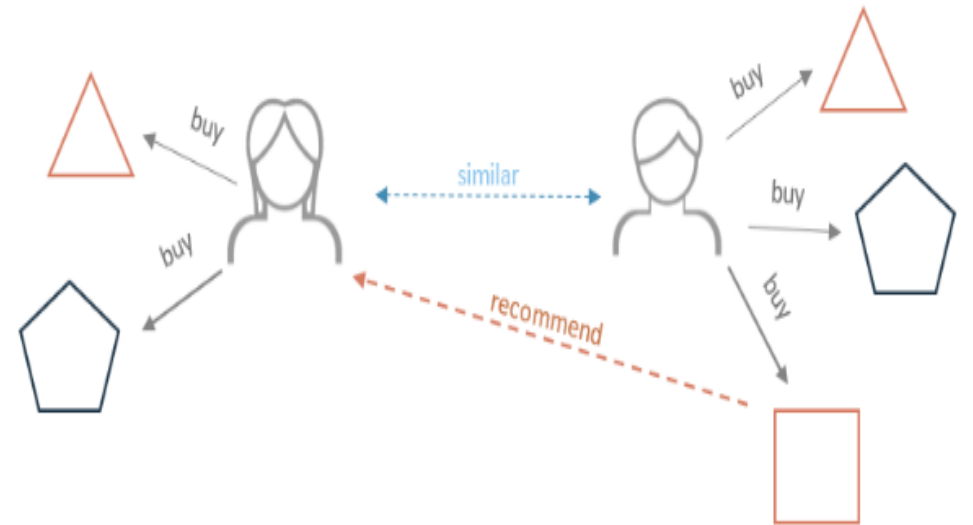
- Providing better and more personalized content to the users.
- More positive feedback from users of these platforms.
- More customer retention for the companies.
- Discovery of new content that the user might not have heard of before but like.
- Increases user engagement in the platform.
- Due to better suggestions spending on the side of the user increases. So revenue of companies increase.
- Acts like a guide to the user through a large collection of content.

Types of Recommendation Engines

- Collaborative Filtering: Collaborative Filtering is a recommendation system technique that collaborates past history of many users to give new recommendations to a specific user.
- Content Based Filtering: Content-based filtering methods are based on a description of the item and a profile of the user's preferences. These algorithms try to recommend items that are similar to those that a user liked in the past, or is examining in the present.
- Multi-Criteria Recommendation Systems: In this system the recommendation is done based on multiple criteria of an item instead of just a single overall preference of an user.
- Mobile Recommender Systems: These are recommendation systems used in mobile devices. The main challenges here are the lack of adequate processing power, noisy data and provision of contextual recommendations. To deal with these problems, most of the processing is done in the cloud and location data is used for contextual awareness.
- Hybrid Recommender Systems: Most recommender systems now combine two or more of the above types to do the recommendations. Ex: Netflix combines both collaborative and content based filtering.
- Session Based Recommendation Systems: These are recommendation engines that do the recommendation based on user interactions in the current session and do not use past data of previous sessions. Ex: YouTube uses such recommendation engines.
- Reinforcement Learning Based Recommendation Systems: In such recommendation systems, the users are the environments while the engines are the agents which act upon these environments in order to receive a reward.

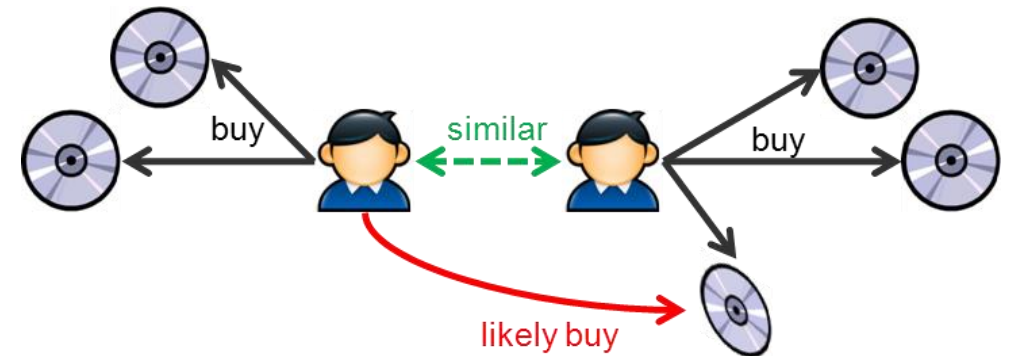
How Collaborative Filters Work

- Collaborative Filtering is a recommendation system technique that collaborates past history of many users to give new recommendations to a specific user.
- The underlying assumption here is that if a person *A* has the same opinion as a person *B* on an issue, *A* is more likely to have *B*'s opinion on a different issue than that of a randomly chosen person.
- The recommender engine has access to a list of past ratings of different users.
- The current user's past ratings are input to the engine.
- The engine then compares these and gives a recommendation to the user.
- Such comparisons can happen in 2 ways:
 1. Memory Based
 2. Model Based



Memory Based

- This method uses a rating system to predict the preferences of one user by taking into consideration the preferences of a similar user, or the 'neighbour'.
- It is of 2 types, user based and item based.
- User based: Here, the existing ratings of an user A is matched with other users with similar ratings for the same items. Then, for the items which have not been rated yet by A, the ratings of these other set of users for these items are considered to predict the likely rating to be given by A for these.
- Item based: In Item-Based Collaborative Filtering, we compare two items and assume them to be similar when one user gives the two items similar ratings. We then predict that user's rating for an item by calculating the weighted average of ratings on most X similar items from this user.



Memory Based

- Here we will use cosine similarity matrix to create the recommender engine.
- Cosine similarity is a mathematical method to find the similarity between 2 items.
- It is the angle in between the spatial distribution vectors of the items and is given by:

$$\cos \theta = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \|\vec{B}\|}$$

Where,

\vec{A} and \vec{B} = Vectors representing the 2 items and are a collection of their numerical features

θ = The angle between the vectors.

- The less the value of the angle the more similar the items are.
- A matrix representing all such cosine similarities between all the items with respect to each other is called a cosine similarity matrix.
- The engine creates such a matrix for the user/item showing the similarities in ratings by different combinations of users for the same movie/different combinations of movies for the same user.
- Then the engine uses KNN algorithm to predict the closer rating (in item based)/closer user (in user based) based on the cosine similarity matrix data.
- In case of user based, the ratings of this closer user is then considered for predicting the rating and for item based the closer item is considered for predicting the rating.

Model Based

- In this method of collaborative filtering recommender systems, different data mining and machine learning algorithms are used to develop a model to predict a user's rating of an unrated item.
- Some examples of these models are Bayesian networks, clustering models, singular value decomposition, probabilistic latent semantic analysis, multiple multiplicative factor, latent Dirichlet allocation and Markov decision process-based models.
- Such models find trends in the ratings being given by users to predict the rating an user will give to an unrated item.
- Here we will use singular value decomposition of matrices.
- It is a decomposition of a rectangular matrix A of dimensions $m \times n$ where,

$$A = USV^T$$

Here,

A = The original $m \times n$ matrix

U = an $m \times m$ orthogonal matrix with eigenvectors of AA^T

S = an $m \times n$ diagonal matrix with eigenvalues of both $A^T A$ and AA^T

V = an $n \times n$ orthogonal matrix with eigenvectors of $A^T A$

- Using SVD we will factorize the matrix of users and ratings into 2 separate matrices of ratings and users.
- These matrices contain vectors products of which model an user's interaction with an item.
- This model will then be used for predicting new ratings.

Project breakdown: Part 1

- Importing of libraries
- Importing of dataset and extracting the ratings data from it
- Dividing the dataset into 2 parts

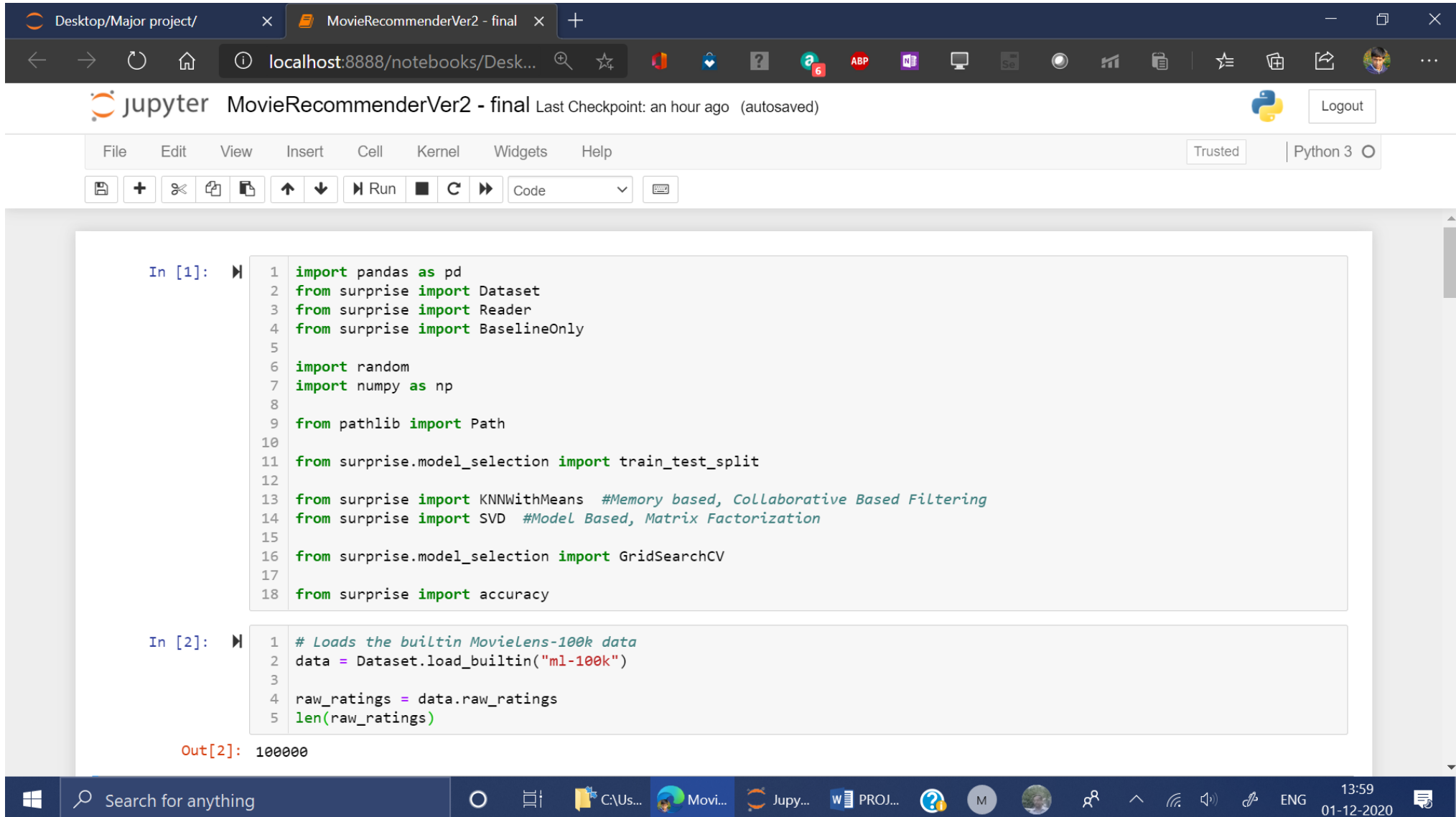
Project breakdown: Part 2

- Setting the hyperparameters for GridSearchCV using KNN algorithm
- Fitting the smaller dataset part in the model
- Finding the best score and set of parameters for the model
- Building the training dataset using the larger part of the dataset and the test dataset using the smaller part.
- Training the model using the training dataset
- Do predictions using the trained model and the test dataset
- Find RMSE, MSE and MAE for Memory based

Project breakdown: Part 3

- Setting the hyperparameters for GridSearchCV using SVD
- Fitting the larger dataset part in the model
- Finding the best score and set of parameters for the model
- Building the full training dataset and the test dataset using the smaller part.
- Training the model using the training dataset
- Do predictions using the trained model and the test dataset
- Find RMSE, MSE and MAE for Matrix factorization based

Implementation



The screenshot displays a Jupyter Notebook titled "MovieRecommenderVer2 - final" running on a local server at `localhost:8888/notebooks/Desk...`. The interface includes a top navigation bar with tabs for "Desktop/Major project/" and "MovieRecommenderVer2 - final". Below the browser address bar, the Jupyter logo and notebook title are visible, along with a "Logout" button. A menu bar contains options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A toolbar below the menu bar includes icons for saving, adding, deleting, and running cells, as well as a dropdown menu currently set to "Code".

The notebook contains two input cells:

```
In [1]: 1 import pandas as pd
        2 from surprise import Dataset
        3 from surprise import Reader
        4 from surprise import BaselineOnly
        5
        6 import random
        7 import numpy as np
        8
        9 from pathlib import Path
        10
        11 from surprise.model_selection import train_test_split
        12
        13 from surprise import KNNWithMeans #Memory based, Collaborative Based Filtering
        14 from surprise import SVD #Model Based, Matrix Factorization
        15
        16 from surprise.model_selection import GridSearchCV
        17
        18 from surprise import accuracy
```

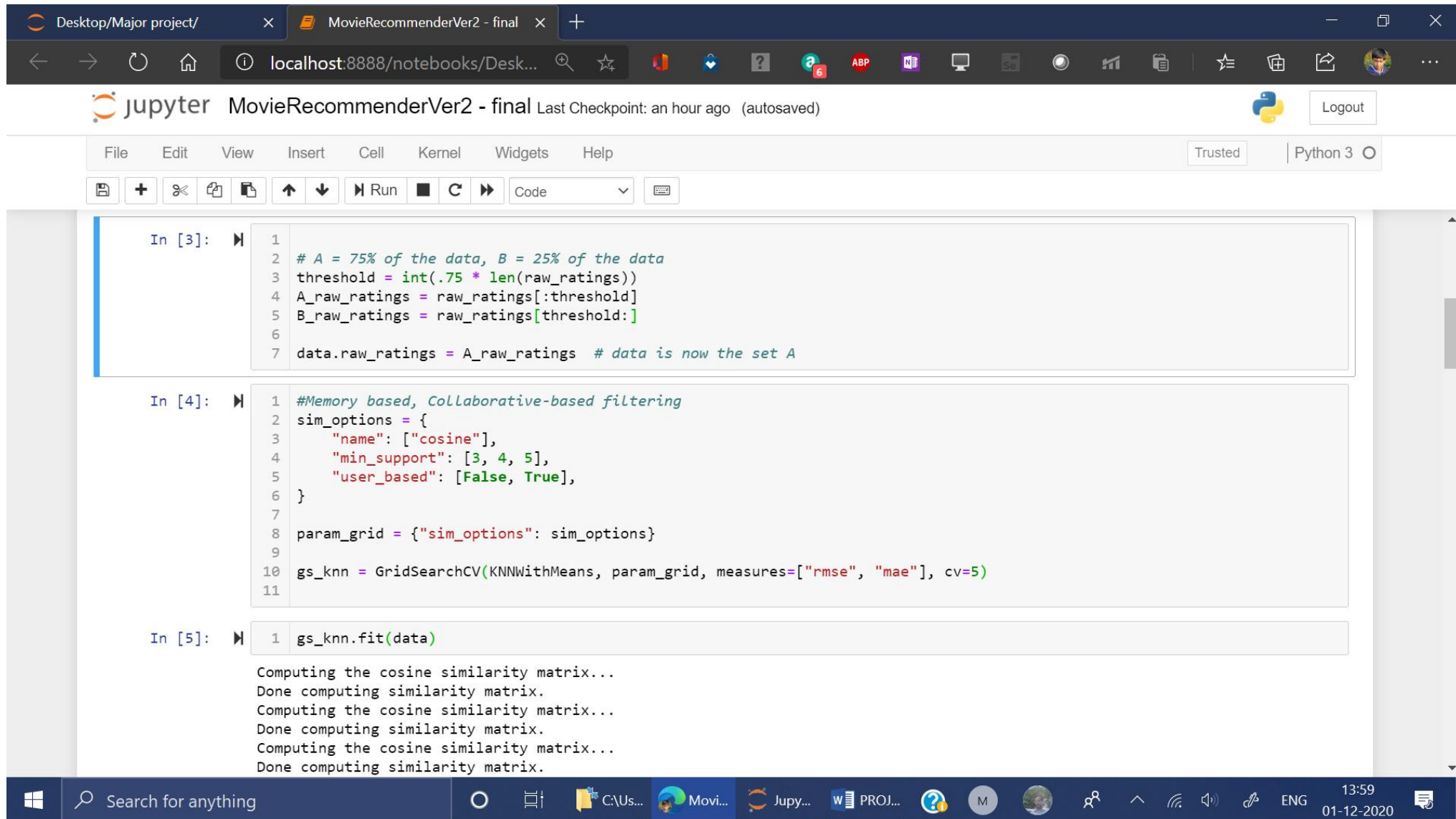
```
In [2]: 1 # Loads the builtin MovieLens-100k data
        2 data = Dataset.load_builtin("ml-100k")
        3
        4 raw_ratings = data.raw_ratings
        5 len(raw_ratings)
```

The output of the second cell is displayed as:

```
Out[2]: 100000
```

The Windows taskbar at the bottom shows the search bar, task view button, and several open applications including "C:\Us...", "Movi...", "Jupy...", and "PROJ...". The system clock indicates the time is 13:59 on 01-12-2020.

Implementation



The screenshot displays a Jupyter Notebook titled "MovieRecommenderVer2 - final" running on a local host. The interface includes a standard menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations and execution. The notebook contains three code cells:

```
In [3]: 1 # A = 75% of the data, B = 25% of the data
2 threshold = int(.75 * len(raw_ratings))
3 A_raw_ratings = raw_ratings[:threshold]
4 B_raw_ratings = raw_ratings[threshold:]
5
6
7 data.raw_ratings = A_raw_ratings # data is now the set A
```

```
In [4]: 1 #Memory based, Collaborative-based filtering
2 sim_options = {
3     "name": ["cosine"],
4     "min_support": [3, 4, 5],
5     "user_based": [False, True],
6 }
7
8 param_grid = {"sim_options": sim_options}
9
10 gs_knn = GridSearchCV(KNNWithMeans, param_grid, measures=["rmse", "mae"], cv=5)
11
```

```
In [5]: 1 gs_knn.fit(data)
```

The output of the third cell shows the progress of the model fitting:

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

The Windows taskbar at the bottom shows the system time as 13:59 on 01-12-2020, along with various application icons.

Implementation

The screenshot displays a Jupyter Notebook titled "MovieRecommenderVer2 - final" running on a local server at localhost:8888. The notebook contains three code cells and a text block.

Code Cell 6:

```
In [6]: 1 print(gs_knn.best_score["rmse"])
        2 print(gs_knn.best_params["rmse"])
        3
```

Output:

```
0.955815409832093
{'sim_options': {'name': 'cosine', 'min_support': 3, 'user_based': False}}
```

Text Block:

Centered-KNN algorithm works best if you go with item-based approach and use cosine similarity as the similarity metric with minimum support 3. Cosine considers common and absolute ratings. However some people like to rate high, even they do not like the item very much. However, some people tend to rate low, even they like the items very much. The traditional cosine similarity does not account for the preference of the user's rating.)

Code Cell 7:

```
In [7]: 1 # We can now use the algorithm that yields the best rmse:
        2 algo_knn = gs_knn.best_estimator['rmse']
        3
        4 trainset = data.build_full_trainset()
        5
        6 testset1 = data.construct_testset(B_raw_ratings) # testset is now the set B
        7
```

Code Cell 8:

```
In [8]: 1
        2 algo_knn.fit(trainset)
```

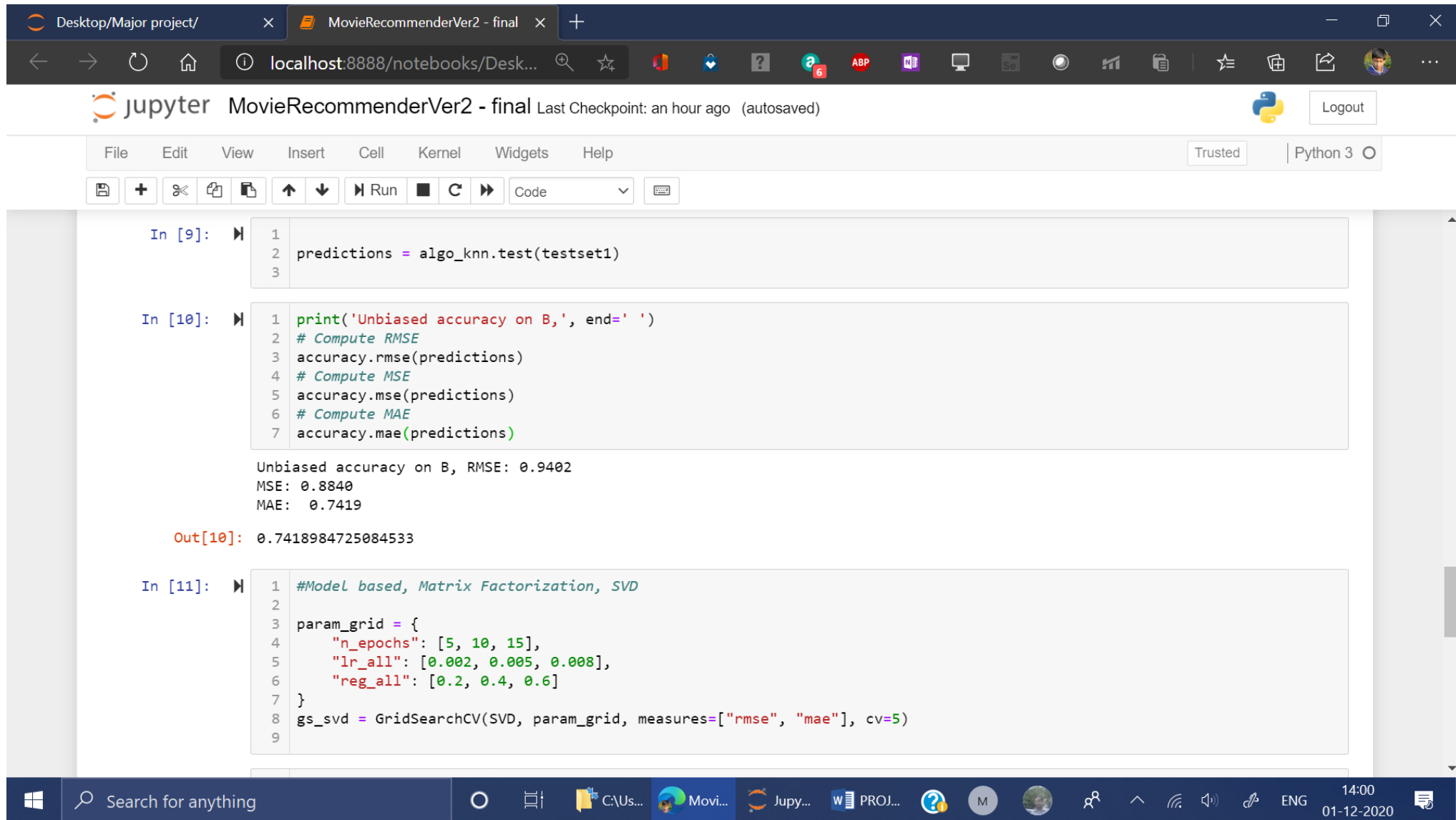
Output:

```
Computing the cosine similarity matrix...
Done computing similarity matrix.

Out[8]: <surprise.prediction_algorithms.knns.KNNWithMeans at 0x24693900ba8>
```

The interface includes a standard menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running code, and other notebook functions. The bottom status bar shows the Windows taskbar with various application icons and the system clock indicating 14:00 on 01-12-2020.

Implementation



The screenshot displays a Jupyter Notebook titled "MovieRecommenderVer2 - final" running on a local host at `localhost:8888/notebooks/Desk...`. The interface includes a standard menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations and code execution. The notebook is set to "Trusted" mode and uses "Python 3".

The code is organized into three input cells:

- In [9]:** A single line of code to test the K-Nearest Neighbors (KNN) algorithm on a test set:

```
1 predictions = algo_knn.test(testset1)
```
- In [10]:** A block of code to calculate and print performance metrics (RMSE, MSE, MAE) for the KNN model:

```
1 print('Unbiased accuracy on B,', end=' ')
2 # Compute RMSE
3 accuracy.rmse(predictions)
4 # Compute MSE
5 accuracy.mse(predictions)
6 # Compute MAE
7 accuracy.mae(predictions)
```

The output of this cell shows the following metrics:

```
Unbiased accuracy on B, RMSE: 0.9402
MSE: 0.8840
MAE: 0.7419
```

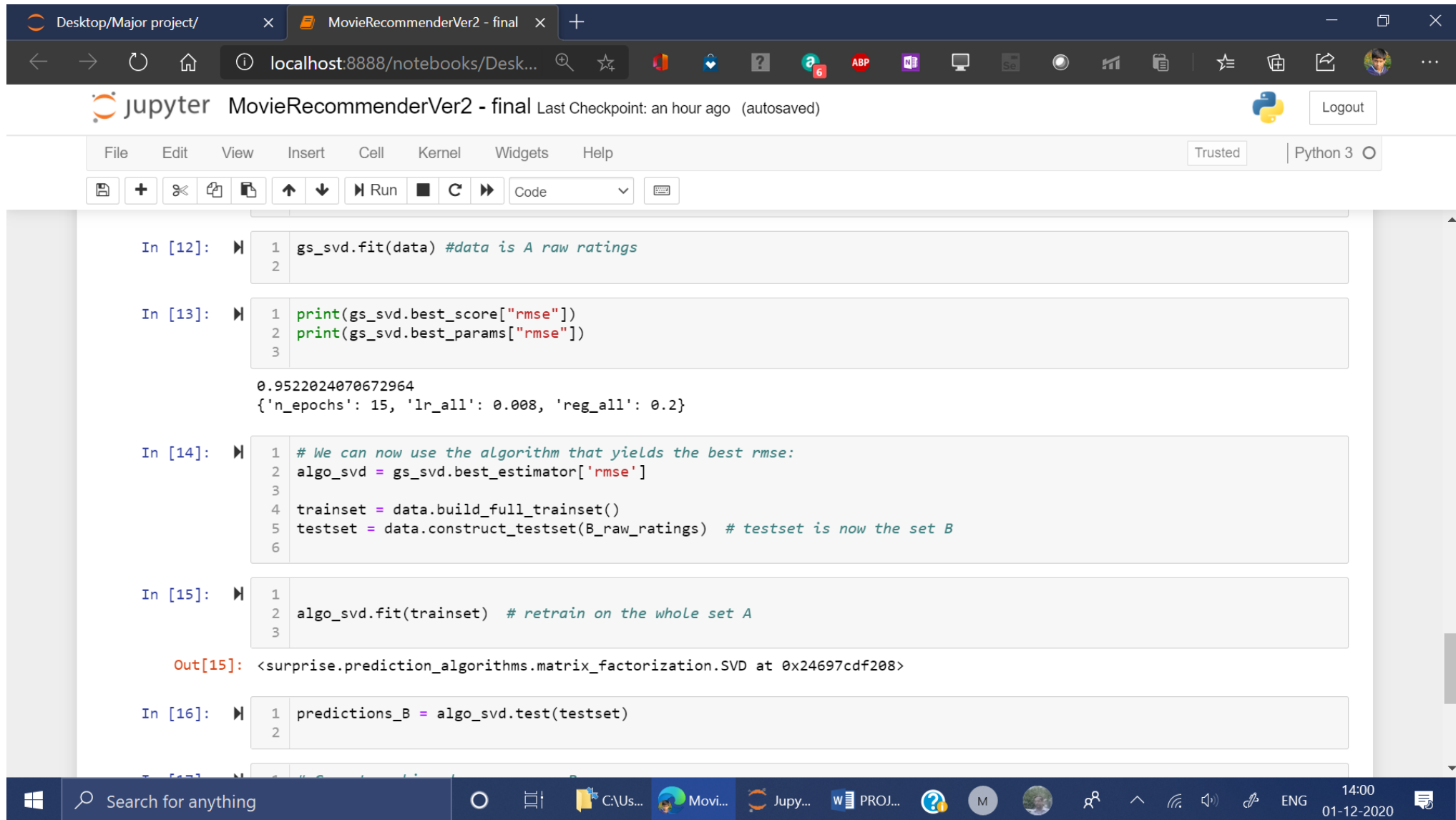
Below the metrics, the output of the last line of code is displayed:

```
Out[10]: 0.7418984725084533
```
- In [11]:** A block of code to set up a Grid Search for a Matrix Factorization model (SVD):

```
1 #Model based, Matrix Factorization, SVD
2
3 param_grid = {
4     "n_epochs": [5, 10, 15],
5     "lr_all": [0.002, 0.005, 0.008],
6     "reg_all": [0.2, 0.4, 0.6]
7 }
8 gs_svd = GridSearchCV(SVD, param_grid, measures=["rmse", "mae"], cv=5)
9
```

The Windows taskbar at the bottom shows the system time as 14:00 on 01-12-2020, along with various application icons including a file explorer, a movie application, and the Jupyter Notebook itself.

Implementation



Desktop/Major project/ × MovieRecommenderVer2 - final × +

localhost:8888/notebooks/Desk... ? 6 ABP N

jupyter MovieRecommenderVer2 - final Last Checkpoint: an hour ago (autosaved) Python 3

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

```
In [12]: 1 gs_svd.fit(data) #data is A raw ratings
          2

In [13]: 1 print(gs_svd.best_score["rmse"])
          2 print(gs_svd.best_params["rmse"])
          3

0.9522024070672964
{'n_epochs': 15, 'lr_all': 0.008, 'reg_all': 0.2}

In [14]: 1 # We can now use the algorithm that yields the best rmse:
          2 algo_svd = gs_svd.best_estimator['rmse']
          3
          4 trainset = data.build_full_trainset()
          5 testset = data.construct_testset(B_raw_ratings) # testset is now the set B
          6

In [15]: 1
          2 algo_svd.fit(trainset) # retrain on the whole set A
          3

Out[15]: <surprise.prediction_algorithms.matrix_factorization.SVD at 0x24697cdf208>

In [16]: 1 predictions_B = algo_svd.test(testset)
          2
```

Search for anything C:\Us... Movi... Jupy... PROJ... ENG 14:00 01-12-2020

Implementation

The screenshot displays a Jupyter Notebook titled "MovieRecommenderVer2 - final" running on a local server at `localhost:8888`. The interface includes a standard menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations and code execution. The notebook content shows the following:

```
Out[15]: <surprise.prediction_algorithms.matrix_factorization.SVD at 0x24697cdf208>
```

```
In [16]: 1 predictions_B = algo_svd.test(testset)
         2
```

```
In [17]: 1 # Compute unbiased accuracy on B
         2 print('Unbiased accuracy on B,', end=' ')
         3 # Compute RMSE
         4 accuracy.rmse(predictions_B)
         5 # Compute MSE
         6 accuracy.mse(predictions_B)
         7 # Compute MAE
         8 accuracy.mae(predictions_B)
```

The output of the code in cell 17 is:

```
Unbiased accuracy on B, RMSE: 0.9431
MSE: 0.8895
MAE: 0.7538
```

```
Out[17]: 0.7538141708989197
```

```
In [ ]: 1
```

The Windows taskbar at the bottom shows the system time as 14:01 on 01-12-2020, along with various application icons including a search bar, file explorer, and the Jupyter application.

Observations

- The following metrics were obtained in the different filters:

Metric	Memory Based	Model Based
MAE	0.7419	0.7538
MSE	0.8840	0.8895
RMSE	0.9402	0.9431

- As we can see, for every metric both the models give almost similar amounts of errors. The memory based filter only performs slightly better which is almost insignificant.
- This shows that both the filters are equally efficient in predicting the user ratings.
- Also, it should be noted that the error is always less than 1.
- This means that the models can predict the user ratings with a maximum error which is less than 1.
- But considering that the ratings are between 1 and 5 and are only integer values, the model should effectively predict the correct rating without any error in most cases.
- The only time there will be an error is when the number of users to predict for is very high and the average rating of all those users are considered for evaluation.

Conclusion

- We thus constructed 2 recommendation engines of collaborative filter type and did predictions using it while checking the performances of these engines
- We found out that the recommendation engines give a maximum error of ≈ 0.75 for MAE and ≈ 0.94 for RMSE i.e. both the errors are < 1 .
- This shows that such recommendation engines are fairly accurate and effective in performing recommendation for users provided the input data given to the engines are cleaned and processed properly.
- Also, the value of these errors can be reduced significantly further by using neural networks for our recommendation engines.
- Hence, such recommendation engines can be used by companies for better predictions and to provide more personalized content for users which will in turn increase customer satisfaction and sales of the company.

Thank You!