IT Fundamentals

Assignment 2

Report of Working With Surprise Lib in Python

Author: Melvin Mokhtari

Student ID: 9831143

Instructor: **Dr. Mohammad Hossein Manshaei**

Date Last Edited: May 26, 2022

The Singular Value Decomposition (SVD) is a linear algebra method that has been widely used in machine learning as a dimensionality reduction technique. SVD is a factorization method for matrices that reduces the number of features in a dataset by changing the number of space dimensions from N to K, where K is less than N. The SVD is used as a collaborative filtering technique in a recommender system. It is structured as a matrix, with each row representing a user and each column representing an item. This matrix's elements are the ratings given to items by users.
KNN is a non-parametric method used for classification. It is also one of the best-known classification algorithms. The principle is that available data is arranged in a space defined by the selected features. When the algorithm gets new data, it will compare the classes of the k closest data points to figure out what the new data class will be.
The significant advantage of the KNN classification is its simplicity; it is also an efficient method. However, computation times can be long with large databases despite their efficiency. Determining the number of neighbors to use (k) requires trial and error. The algorithm is flawed with outliers that can strongly impact its efficiency.
To put it succinctly, Generally, SVD provides a more accurate prediction compared to KNN. In most cases, KNN is better in data sets with a low missing data proportion. Furthermore, SVD performs better in massively sized datasets.
The SVD does suffer from a drawback: it does not perform well when there are missing values in the rating matrix. To overcome this, you have to preprocess the data. So, KNN algorithms are responsible for identifying the K-nearest neighbors of a given user. They work well in this situation because, based on their ratings, you could use these algorithms to fill in the missing ratings for the user in question.
I also have to go over the CV algorithm. Cross-validation is a resampling method used to test machine learning models with a small amount of data. The procedure has a single parameter called k, which refers to the number of groups that a given data sample is to be split into.
A K-Fold CV is where a given data set is split into a K number of sections or folds where each fold is used as a testing set at some point. Let us take the scenario of four-fold cross-validation (K=4). Here, the data set is split into four folds. The first fold is used to test the model in the first iteration, and the rest are used to train the model. The second fold is used as the testing set in the second iteration, while the rest serves as the training set. This process is repeated until each of the four folds has been used as the testing set.
On the other hand, Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) are metrics used to evaluate a regression model. These metrics tell us how accurate our predictions are and the amount of deviation from the actual values.
Both MAE and RMSE express the average model prediction error in units of the variable of interest. Both metrics can range from 0 to infinity and are indifferent to the direction of errors. They are negatively-oriented scores, which means lower values are better. The RMSE will always be larger or equal to the MAE; the greater the difference between them, the greater the variance in the individual errors in the sample. If the RMSE = MAE, then all the errors are of the same magnitude.
We also have STD on the output, the standard deviation, and the mean, which is equal to the average. In math, the mean is the average of all the given values. It means that all the given values are divided by the total number of values given. Simultaneously, the standard deviation is regarded as a measure of the dispersion of data values from the mean. It means how far the data values are spread out from the mean value. For example, if you calculated the STD of scores from a class of students of similar ability, you would expect it to be below because the scores would all be close to the mean. On the other hand, you would expect the STD of scores from a mixed-ability class to be higher. If these calculations did not conform to expectations, you would want to look more closely at the data to check for inaccuracies.
Well, I have created a test set with the following specifications:

- Only our test set values for items or users are on that set.
- Those test set values are hidden from the training data.
- I have trained our model only on the user-item pairs where I have data.

Moreover, as a result, I have worked on this test set to evaluate the algorithms. So now I have enough information about these words and expressions in the output of my codes.

Let's look at the output of my codes now.
In the first, it is evident that I randomly shuffled the dataset into four sets of identical sizes (this is usually implemented by shuffling the data array and then splitting it into four). Then I train on three of them and validate on one, then train on the remaining three and validate on another.
For example, fold one has less RMSE than fold four, indicating that the data in fold four is not as concentrated as the data in fold one, implying a better model to predict the data more accurately than other folds. The lower the RMSE, the better a given model can "fit" a dataset, so fold one is the best on the RMSE scale, but RMSE is not the only factor to consider!
MAE is also crucial in the evaluation process. The lower the MAE value, the better the model, and zero means it is perfect!
This value is higher for fold one than for the other folds, so we conclude that fold one has a better model for our dataset by taking these two factors and the time elapsed into account.

HOWEVER...

But it's important to remember that folds won't be judged on their own. Even if fold one is a better model, the average of these rates will be taken into account when comparing two algorithms like SVD and KNN.
In this scenario, the SVD algorithm's average RMSE and MAE are lower than the RMSE and MAE of KNN, indicating that the SVD algorithm is a better algorithm for our large dataset. Furthermore, comparing the two will yield the same result.
Then we had to divide our dataset into train set and test set parts, which we accomplished thoroughly by adding 0.25 to our function; as a result, 25% of our data will be randomly assigned to the test set each time. Then we computed RMSE to determine whether our prediction model was adequate. As previously stated, the lower the RMSE, the better a given model can "fit" a dataset.
The next step is to forecast all of the data from the test set. Here, we have two options:

1. Using the algo.test (testset) command
2. Utilizing the predict () method

As a result, I went with the second option. To use all of the test set data, we must create a nested loop and enter all of the uid and iid of users and items while paying attention to the uid and iid string format.
Then we'll see a long list of users and items that might match, followed by a message indicating whether this prediction is correct or incorrect. At the same time, an estimation rate describes how likely these matches are.
If we want to compare these two algorithms based on the results of these tests, we contend that the SVD algorithm is faster, more accurate, and well-suited to large databases. KNN algorithms, on the other hand, are better suited for smaller databases because they may take longer to analyze test sets.

- Please check the comment section of this assignment on "Yekta" for the link of the video of this report's explanations.
- The following pages contain original codes and screenshots of the execution results:

1 Codes

1.1 SVD - Part 1

```python
from surprise import SVD
from surprise import Dataset
from surprise.model_selection import cross_validate

# Load the movielens-100k dataset (download it if needed),
data = Dataset.load_builtin('ml-100k')

# We'll use the famous SVD algorithm.
algo = SVD()

# Run 4-fold cross-validation and print results.
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=4, verbose=True)
```

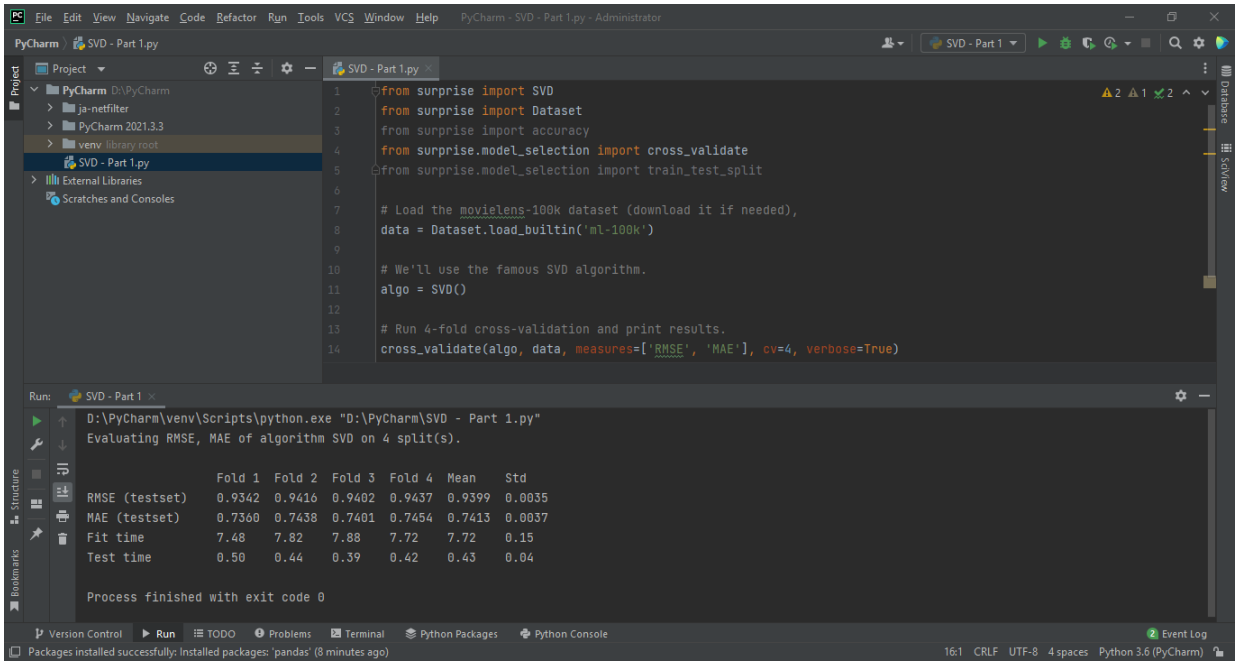

Figure 1: Execution result

1.2 SVD - Part 2

```python
from surprise import SVD
from surprise import Dataset
from surprise import accuracy
from surprise.model_selection import train_test_split

# Load the movielens-100k dataset (download it if needed),
data = Dataset.load_builtin('ml-100k')

# sample random trainset and testset
# test set is made of 25\% of the ratings.
trainset, testset = train_test_split(data, test_size=.25)

# We'll use the famous SVD algorithm.
algo = SVD()

# Train the algorithm on the trainset, and predict ratings for the testset
predictions = algo.fit(trainset).test(testset)

# Then compute RMSE
accuracy.rmse(predictions)
```

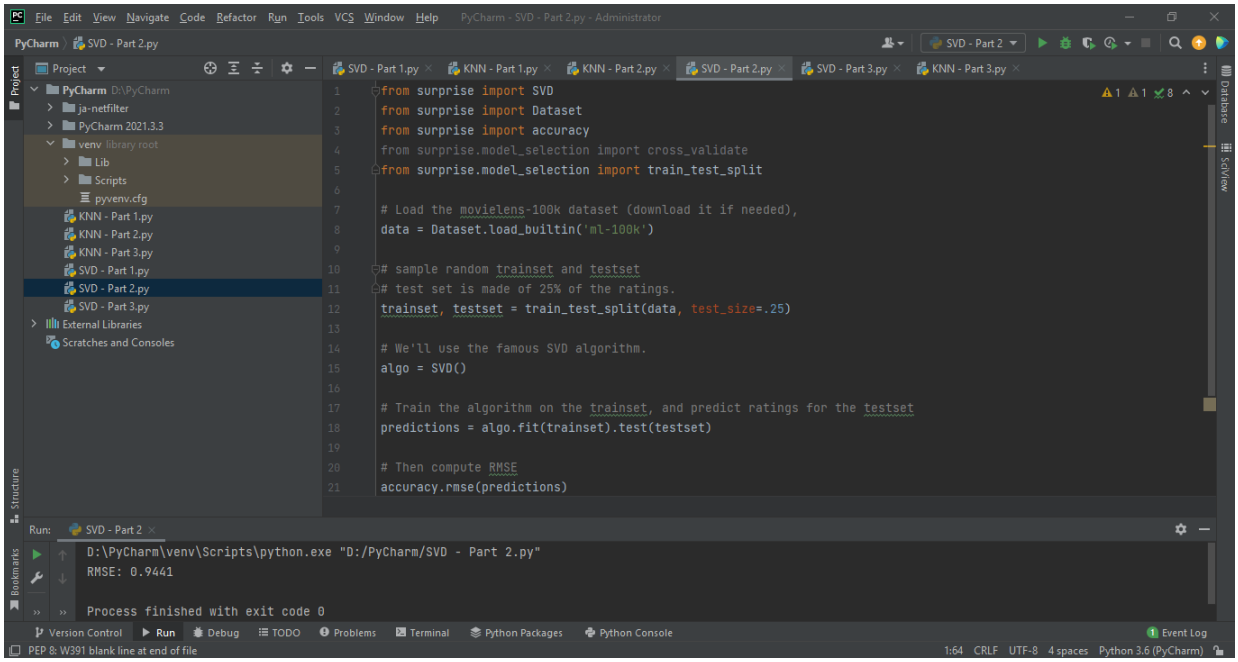

Figure 2: Execution result

1.3 SVD - Part 3

```
from surprise import SVD
from surprise import Dataset
from surprise.model_selection import train_test_split

# Load the movielens-100k dataset (download it if needed),
data = Dataset.load_builtin('ml-100k')

# sample random trainset and testset
# test set is made of 25% of the ratings.
trainset, testset = train_test_split(data, test_size=.25)

# We'll use the famous SVD algorithm.
algo = SVD()

# Train the algorithm on the trainset and the testset
algo.fit(trainset)

#predict ratings by directly calling the predict() method
# raw user id (as in the ratings file). They are **strings**!
# raw item id (as in the ratings file). They are **strings**!
#The full dataset contains 100000 ratings by 943 users on 1682 items.
testset = trainset.build_anti_testset()
predictions = algo.test(testset)
for uid in predictions:
for iid in predictions:
pred = algo.predict(str(uid), str(iid), r_ui=4, verbose=True)
```

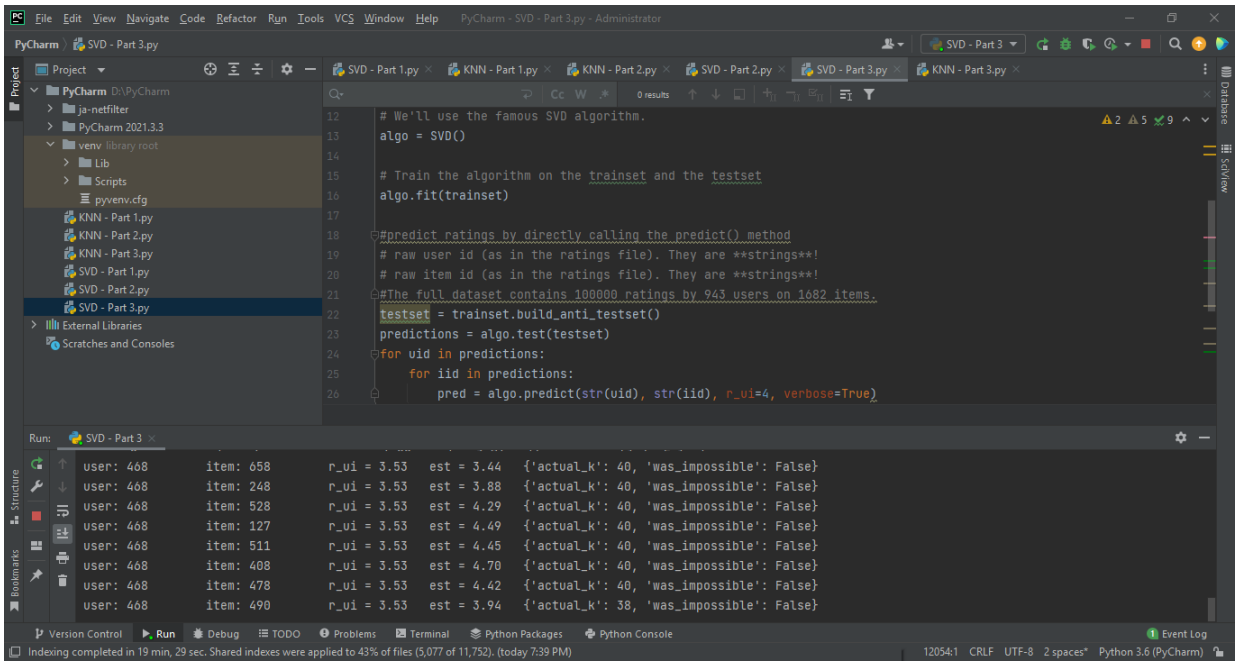

Figure 3: Execution result

1.4 KNN - Part 1

```python
from surprise import KNNBasic
from surprise import Dataset
from surprise.model_selection import cross_validate

# Load the movielens-100k dataset (download it if needed),
data = Dataset.load_builtin('ml-100k')

# We'll use the famous KNNBasic algorithm.
algo = KNNBasic()

# Run 4-fold cross-validation and print results.
cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=4, verbose=True)
```

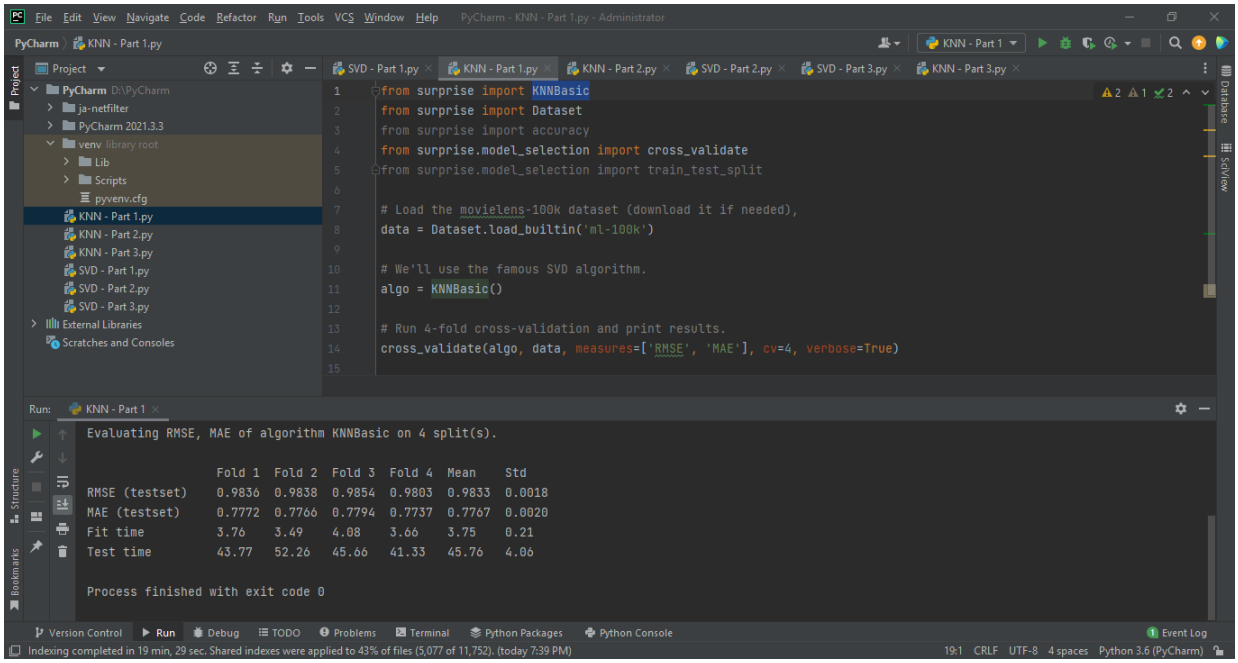

Figure 4: Execution result

1.5 KNN - Part 2

```python
from surprise import KNNBasic
from surprise import Dataset
from surprise import accuracy
from surprise.model_selection import train_test_split

# Load the movielens-100k dataset (download it if needed),
data = Dataset.load_builtin('ml-100k')

# sample random trainset and testset
# test set is made of 25\% of the ratings.
trainset, testset = train_test_split(data, test_size=.25)

# We'll use the famous KNNBasic algorithm.
algo = KNNBasic()

# Train the algorithm on the trainset, and predict ratings for the testset
predictions = algo.fit(trainset).test(testset)

# Then compute RMSE
accuracy.rmse(predictions)
```

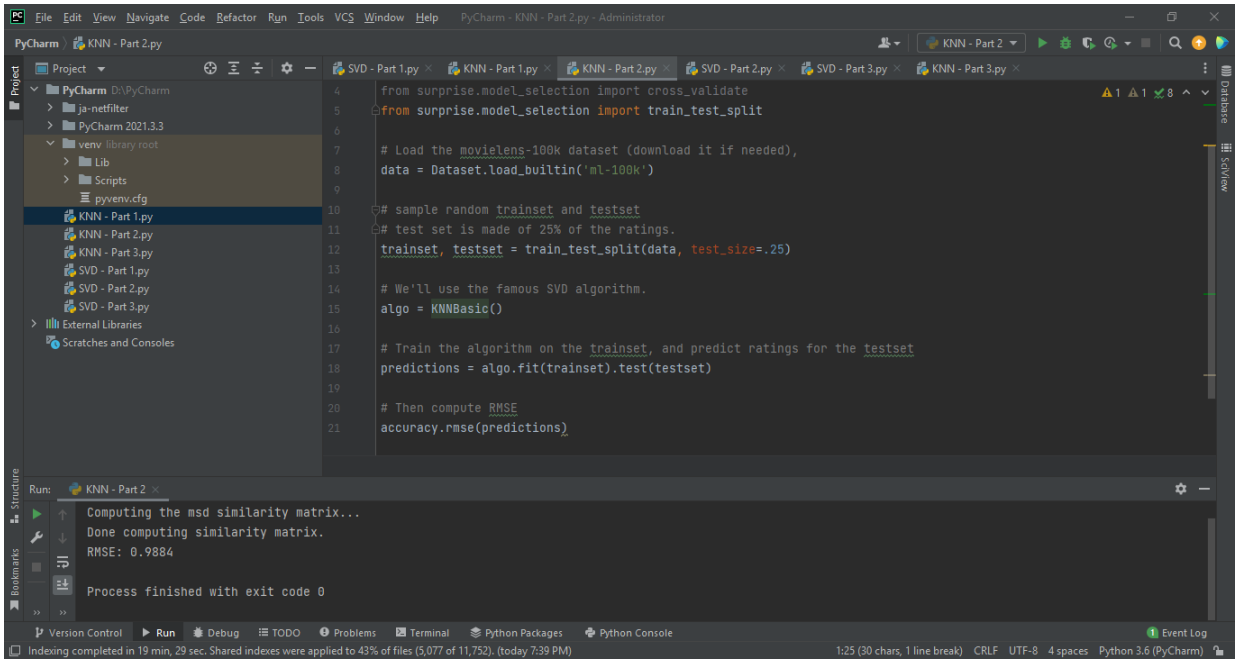

Figure 5: Execution result

1.6 KNN - Part 3

```
from surprise import KNNBasic
from surprise import Dataset
from surprise.model_selection import train_test_split

# Load the movielens-100k dataset (download it if needed),
data = Dataset.load_builtin('ml-100k')

# sample random trainset and testset
# test set is made of 25\% of the ratings.
trainset, testset = train_test_split(data, test_size=.25)

# We'll use the famous KNNBasic algorithm.
algo = KNNBasic()

# Train the algorithm on the trainset and the testset
algo.fit(trainset)

#predict ratings by directly calling the predict() method
# raw user id (as in the ratings file). They are **strings**!
# raw item id (as in the ratings file). They are **strings**!
#The full dataset contains 100000 ratings by 943 users on 1682 items.
testset = trainset.build_anti_testset()
predictions = algo.test(testset)
for uid in predictions:
for iid in predictions:
pred = algo.predict(str(uid), str(iid), r_ui=4, verbose=True)
```

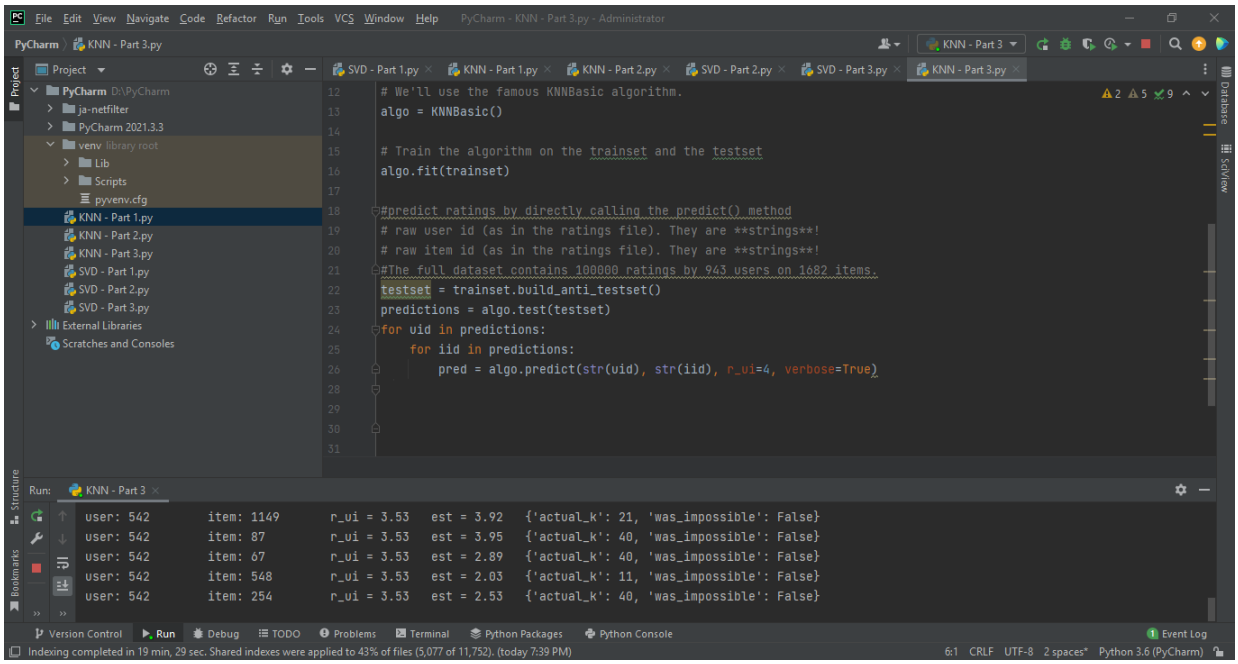

Figure 6: Execution result