Department of Computer Science and Mathematics

CSC 599 – Capstone Project

Dr. Nadine Abbas

Recommender Systems for E-commerce

Patrick Nohra - 201800121

Contents

Abstract

E-commerce has been booming since the 2010s, and now that practically everyone in the world has access to the internet, everyone can purchase online. However, we may improve his experience by recommending goods to him. In this project, machine learning will be used to create a recommender system for an e-commerce website.

Intro

With the fast expansion of e-commerce, an increasing number of people have flocked to e-commerce platforms. E-commerce platforms frequently propose items that consumers may enjoy or need in order to deliver a better purchasing experience.

A Recommender System (RS) is an engine that suggests previously unknown items that the user might be interested in and supports them in making decisions. When users accomplish such tasks online, their lives become easier and more convenient. However, the user may become confused in the large amount of information supplied by the programs, wasting substantial time. Recommender systems enhances application performance and attracts users, resulting in application success and making the recommender system innovative. A recommendation system is a type of system that collects and analyzes consumer product data to propose things that may be satisfying to users. This allows for the establishment of a match between items and customers.

The recommendation system is critical in the e-commerce platform.

Methodology

For this project, we will be implementing a popularity-based recommender system, and collaborative filtering.

A recommendation system based on popularity follows the trend. It simply employs items that are currently popular. For example, if a product is often purchased by every new user, it may propose that item to the person who has just joined up. The issue with popularity-based recommendation systems is that personalization is not accessible, which means that even if you know the user's behavior, you cannot propose goods properly.

On the other hand, collaborative filtering is more commonly used in recommender systems. These techniques are designed to fill in the gaps in a user-item association matrix. We will employ the collaborative filtering (CF) method. The premise behind CF is that the greatest suggestions come from individuals who share similar tastes.

In other words, it predicts how someone would evaluate an item based on prior item evaluations from like people.

Memory-based and model-based techniques are the two subcategories of collaborative filtering.

Model-based CF technique builds model by using existing relationships of users and items and learns the model to predict accurately. Some popular model-based techniques are Singular Value Decomposition (SVD), Matrix Factorization (MF), clustering, Bayesian networks etc.

Memory based CF technique first finds a group of users with similar tastes by using similarity measures such as Pearson correlation, adjusted cosine similarities etc. and then recommends items based on the choices of the similar users.

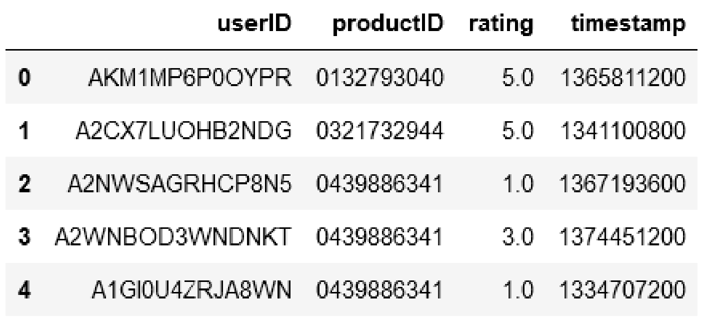
Libraries

The libraries used are:

- Numpy: faster matrix related computations and offers a lot of methods to get meaningful information from a dataset.
- Pandas: read and manipulate the dataset
- Seaborn: visualize data
- Sklearn: contains method used for prediction metrics. Also provides important learning methods.
- Surprise: an extended module from sklearn to work specifically with recommender systems.

Dataset

The dataset that will be used is the “amazon review dataset”.



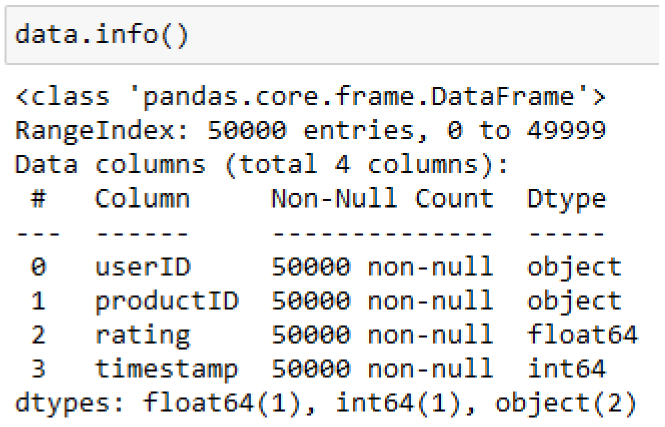
	userID	productID	rating	timestamp
0	AKM1MP6P0OYPR	0132793040	5.0	1365811200
1	A2CX7LUOHB2NDG	0321732944	5.0	1341100800
2	A2NWSAGRHCP8N5	0439886341	1.0	1367193600
3	A2WNBOD3WNDNKT	0439886341	3.0	1374451200
4	A1GI0U4ZRJA8WN	0439886341	1.0	1334707200

Figure 1 Dataset preview

The dataset consists of 4 columns and 7824482 rows. The columns are as follow:

- ❖ userId: Each user is assigned a unique id.
- ❖ productId: Each product is assigned a unique id.
- ❖ Rating: The related user's rating of the associated product.
- ❖ timestamp: The rating's time.

Due to the limitations of this project, we will limit the number of rows to 50,000 and drop the timestamp column to save more RAM.



```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   userID     50000 non-null  object 
 1   productID  50000 non-null  object 
 2   rating     50000 non-null  float64
 3   timestamp  50000 non-null  int64  
dtypes: float64(1), int64(1), object(2)
```

Regarding data pre-processing, the data we have already met all the preprocessing criterion. There isn’t any null value that we have to deal with, and all the columns are formatted as we need them.

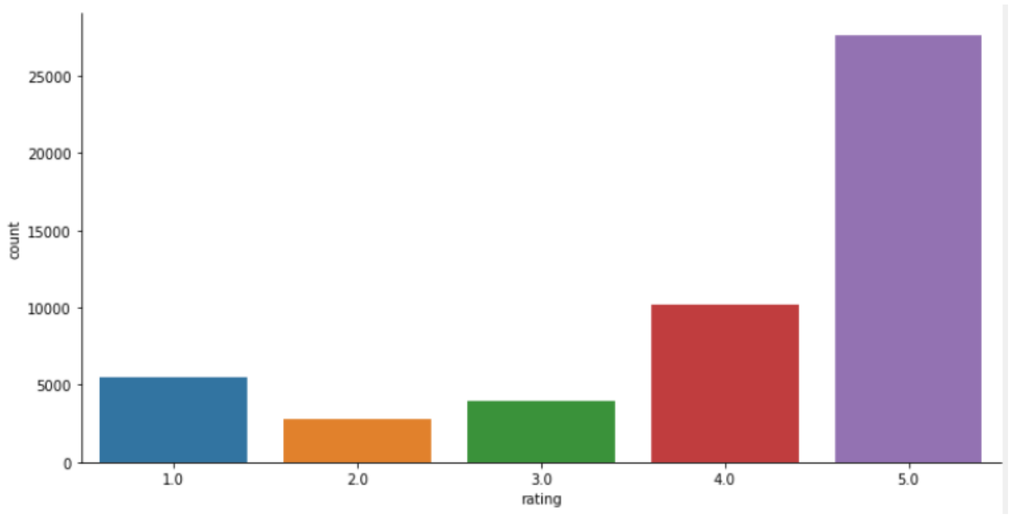


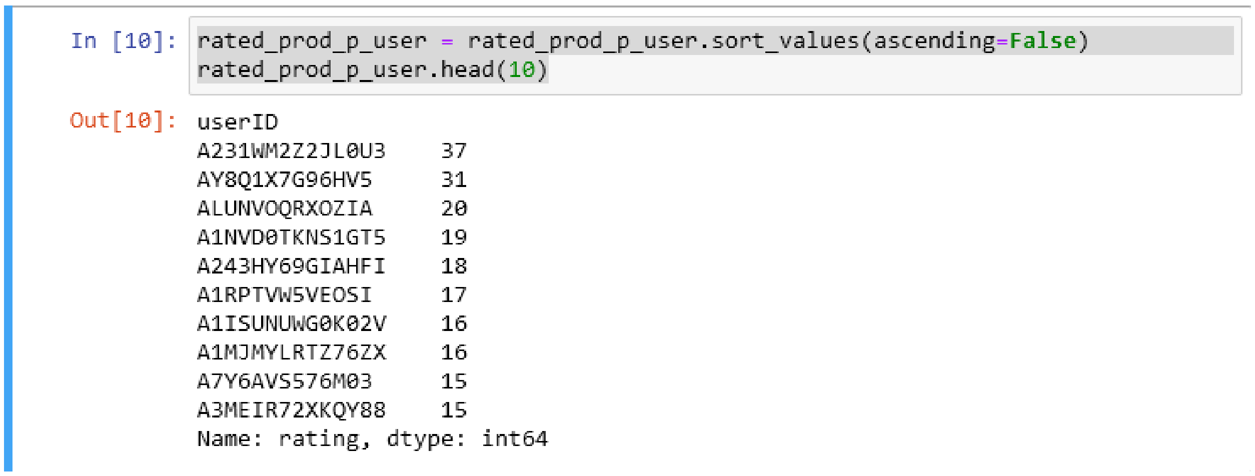
Figure 2 Data distribution

Figure 2 shows the distribution of the data, and it can be seen that the majority of the items are rated 5, and that the number of items rated 5 is five times greater than the number of items rated 1. In addition, the dataset is constituted of 46554 unique users and 3446 unique products.

We should also have an idea about the users who submitted the most reviews.

```
rated_prod_p_user = data.groupby(by='userID')['rating'].count()
# print(rated_prod_p_user.head)
rated_prod_p_user.describe()
```

```
count    46554.000000
mean         1.074022
std          0.520846
min          1.000000
25%          1.000000
50%          1.000000
75%          1.000000
max         37.000000
Name: rating, dtype: float64
```



```
In [10]: rated_prod_p_user = rated_prod_p_user.sort_values(ascending=False)
         rated_prod_p_user.head(10)

Out[10]: userID
         A231WM2Z2JL0U3    37
         AY8Q1X7G96HV5     31
         ALUNVOQRXOZIA     20
         A1NVD0TKNS1GT5    19
         A243HY69GIAHFI    18
         A1RPTVW5VEOSI     17
         A1ISUNUWG0K02V    16
         A1MJMYLRTZ76ZX    16
         A7Y6AVS576M03     15
         A3MEIR72XKQY88    15
         Name: rating, dtype: int64
```

Figure 3 Rated products per users

Figure 3 shows that the highest number of rated products per user is 37, however the mean count of ratings is only 1.07 reviews per person.

After analyzing the dataset, we can move to the implementation of the recommender systems.

Popularity-based recommendations

We first start by creating a new matrix containing the items that have more than 50 reviews.

```
popular =  data.groupby('productID').filter(lambda x:x['rating'].count()>50)
```

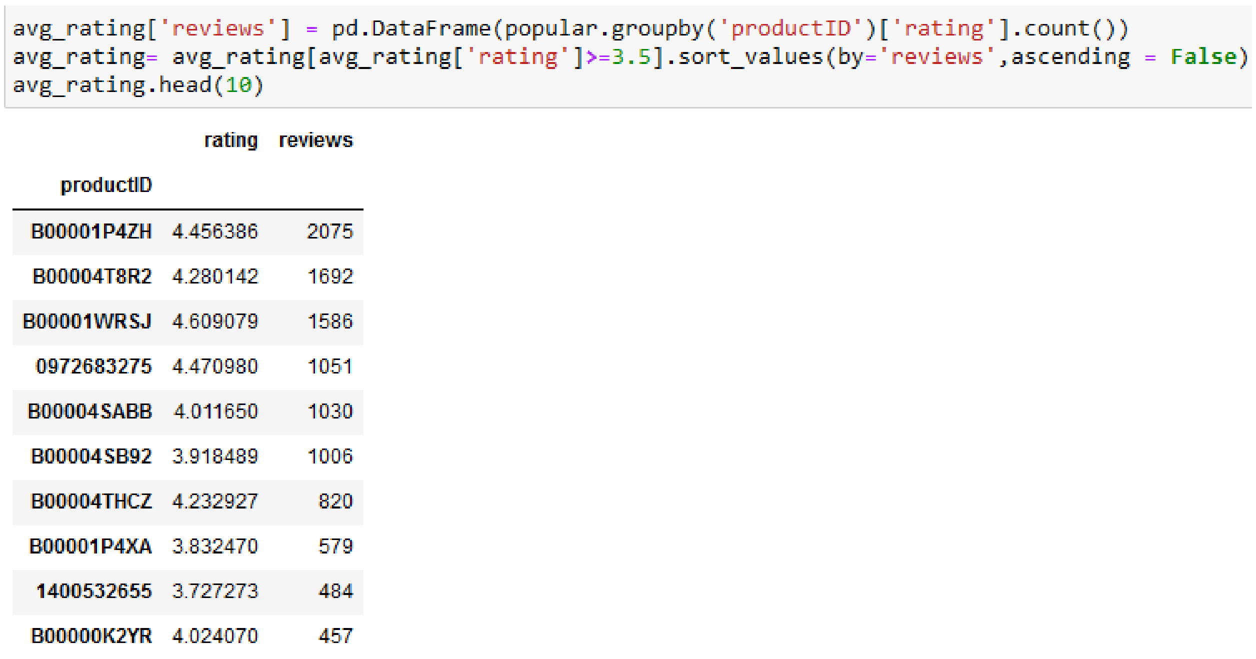
We then created another matrix containing the number of ratings per item.

```
ratings_per_item = popular.groupby(by= 'productID')['rating'].count().sort_values(ascending = False)
```

We then also get the average rating of each product and store them in a new dataframe.

```
avg_rating =pd.DataFrame(popular.groupby(by='productID')['rating'].mean())
```

After getting the average rating of each product and storing it in a matrix, we finally add the total number of reviews, remove all items that have a mean rating lower than 3.5, and sort the dataframe in a descending order by the total number of reviews per item.



```
avg_rating['reviews'] = pd.DataFrame(popular.groupby('productID')['rating'].count())
avg_rating= avg_rating[avg_rating['rating']>=3.5].sort_values(by='reviews',ascending = False)
avg_rating.head(10)
```

productID	rating	reviews
B00001P4ZH	4.456386	2075
B00004T8R2	4.280142	1692
B00001WRSJ	4.609079	1586
0972683275	4.470980	1051
B00004SABB	4.011650	1030
B00004SB92	3.918489	1006
B00004THCZ	4.232927	820
B00001P4XA	3.832470	579
1400532655	3.727273	484
B00000K2YR	4.024070	457

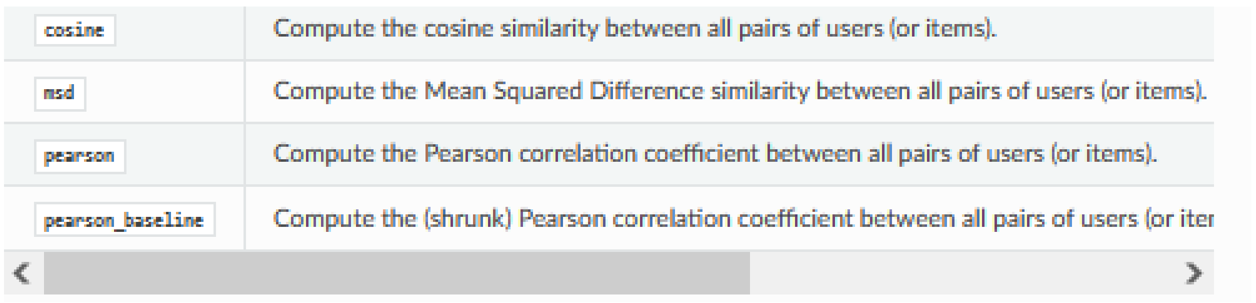
Figure 4 Recommended items based on the popularity-based model

As seen in figure 4, these are the top 10 items that will be recommended for a user built by the popularity-based model.

Collaborative Filtering

I - Memory based Collaborative Filtering

In the model-based approach, we will use the KNN with means algorithm to predict the rating of the item. KNN with means is a basic collaborative filtering algorithm, taking into account the mean ratings of each user. We first split the data into training and testing data, 75% and 25% respectively and then train the model on the training data and get output based on the testing data. Before going through the training step, there should be a basic understanding of the similarity measures.



`cosine`	Compute the cosine similarity between all pairs of users (or items).
`msd`	Compute the Mean Squared Difference similarity between all pairs of users (or items).
`pearson`	Compute the Pearson correlation coefficient between all pairs of users (or items).
`pearson_baseline`	Compute the (shrunk) Pearson correlation coefficient between all pairs of users (or iter

The most commonly used similarity measures are shown in the figure above. The chosen measure in this project is the Pearson baseline. When only a few ratings are available, the shrinkage parameter helps in avoiding overfitting .

The Pearson-baseline correlation coefficient is calculated as follows:

$$\text{pearson\_baseline\_sim}(u,v) = \hat{\rho}_{uv} = \frac{\sum\limits_{i \in I_{uv}} (r_{ui} - b_{ui}) \cdot (r_{vi} - b_{vi})}{\sqrt{\sum\limits_{i \in I_{uv}} (r_{ui} - b_{ui})^2} \cdot \sqrt{\sum\limits_{i \in I_{uv}} (r_{vi} - b_{vi})^2}}$$

or

$$\text{pearson\_baseline\_sim}(i,j) = \hat{\rho}_{ij} = \frac{\sum\limits_{u \in U_{ij}} (r_{ui} - b_{ui}) \cdot (r_{uj} - b_{uj})}{\sqrt{\sum\limits_{u \in U_{ij}} (r_{ui} - b_{ui})^2} \cdot \sqrt{\sum\limits_{u \in U_{ij}} (r_{uj} - b_{uj})^2}}$$

The Pearson-baseline correlation coefficient is then defined as follows:

$$\text{pearson\_baseline\_shrunk\_sim}(u,v) = \frac{|I_{uv}| - 1}{|I_{uv}| - 1 + \text{shrinkage}} \cdot \hat{\rho}_{uv}$$

$$\text{pearson\_baseline\_shrunk\_sim}(i,j) = \frac{|U_{ij}| - 1}{|U_{ij}| - 1 + \text{shrinkage}} \cdot \hat{\rho}_{ij}$$

The shrinkage parameter that will be used is 100.

After training the model, the predictions should look like this:

Prediction(uid='A2XNOB1T796Y6B', iid='1400532655', r_ui=4.0, est=4.123157620946766
Our model estimated a rating of 4.123 for this item.
A good metric for recommender system evaluation is the root mean squared error (RMSE). The RMSE formula is :

$$\text{RMSE} = \sqrt{\frac{1}{|\hat{R}|} \sum_{\hat{r}_{ui} \in \hat{R}} (r_{ui} - \hat{r}_{ui})^2}.$$

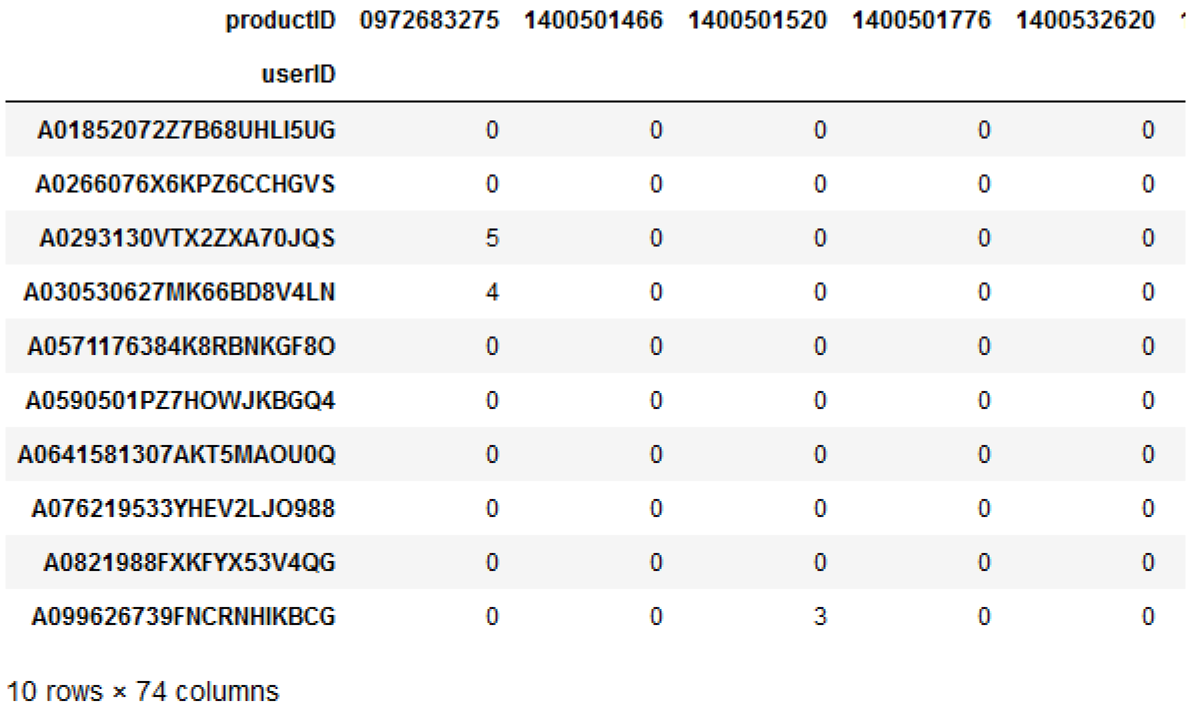
After testing the model, the final RMSE value is 1.2781.

II - Model Based Collaborative Filtering

In the model-based approach, we will be using TruncatedSVD. This transformer reduces linear dimensionality via truncated singular value decomposition (SVD). This estimator, unlike PCA, does not center the data before generating the singular value decomposition.

This means it can work effectively with sparse matrices, which is our case.

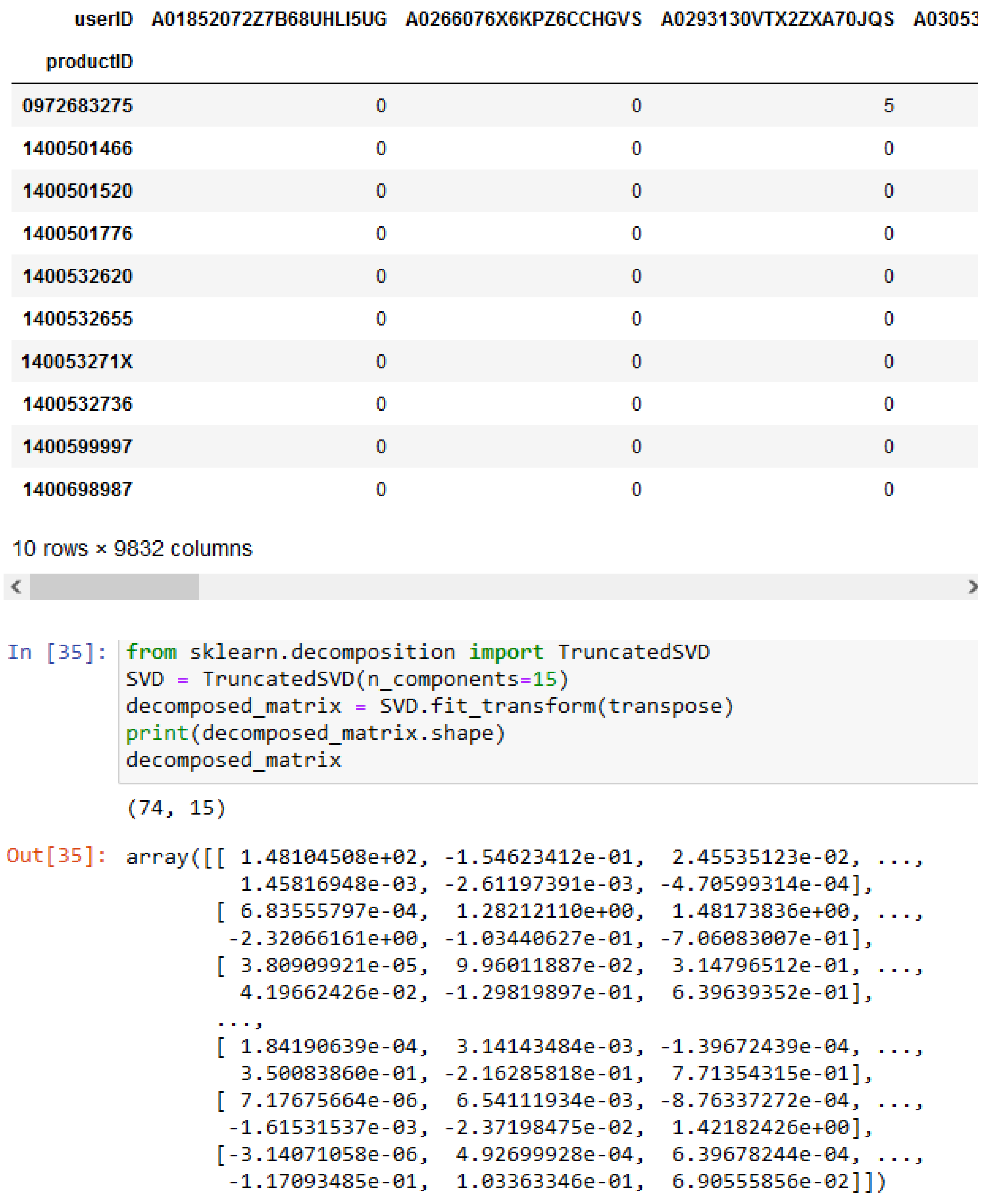
We first need to convert our dataframe into a table, such that the columns represent the items, the rows represent the costumers, and the value at ($item_i$, $costumer_j$) is the rating the costumer set for the specific item.



productID userID	0972683275	1400501466	1400501520	1400501776	1400532620
A01852072Z7B68UHLI5UG	0	0	0	0	0
A0266076X6KPZ6CCHGVS	0	0	0	0	0
A0293130VTX2ZXA70JQS	5	0	0	0	0
A030530627MK66BD8V4LN	4	0	0	0	0
A0571176384K8RBNKGF8O	0	0	0	0	0
A0590501PZ7HOWJKBGQ4	0	0	0	0	0
A0641581307AKT5MAOU0Q	0	0	0	0	0
A076219533YHEV2LJO988	0	0	0	0	0
A0821988FXKFYX53V4QG	0	0	0	0	0
A099626739FNCRNHIKBCG	0	0	3	0	0

10 rows × 74 columns

If the user did not rate the item, the rating is set to 0. We then transposed the matrix so we could pass it as an argument to the truncatedSVD and reduce the number of users to 15.



userID	A01852072Z7B68UHLI5UG	A0266076X6KPZ6CCHGVS	A0293130VTX2ZXA70JQS	A03053
productID				
0972683275	0	0	5	
1400501466	0	0	0	
1400501520	0	0	0	
1400501776	0	0	0	
1400532620	0	0	0	
1400532655	0	0	0	
140053271X	0	0	0	
1400532736	0	0	0	
1400599997	0	0	0	
1400698987	0	0	0	

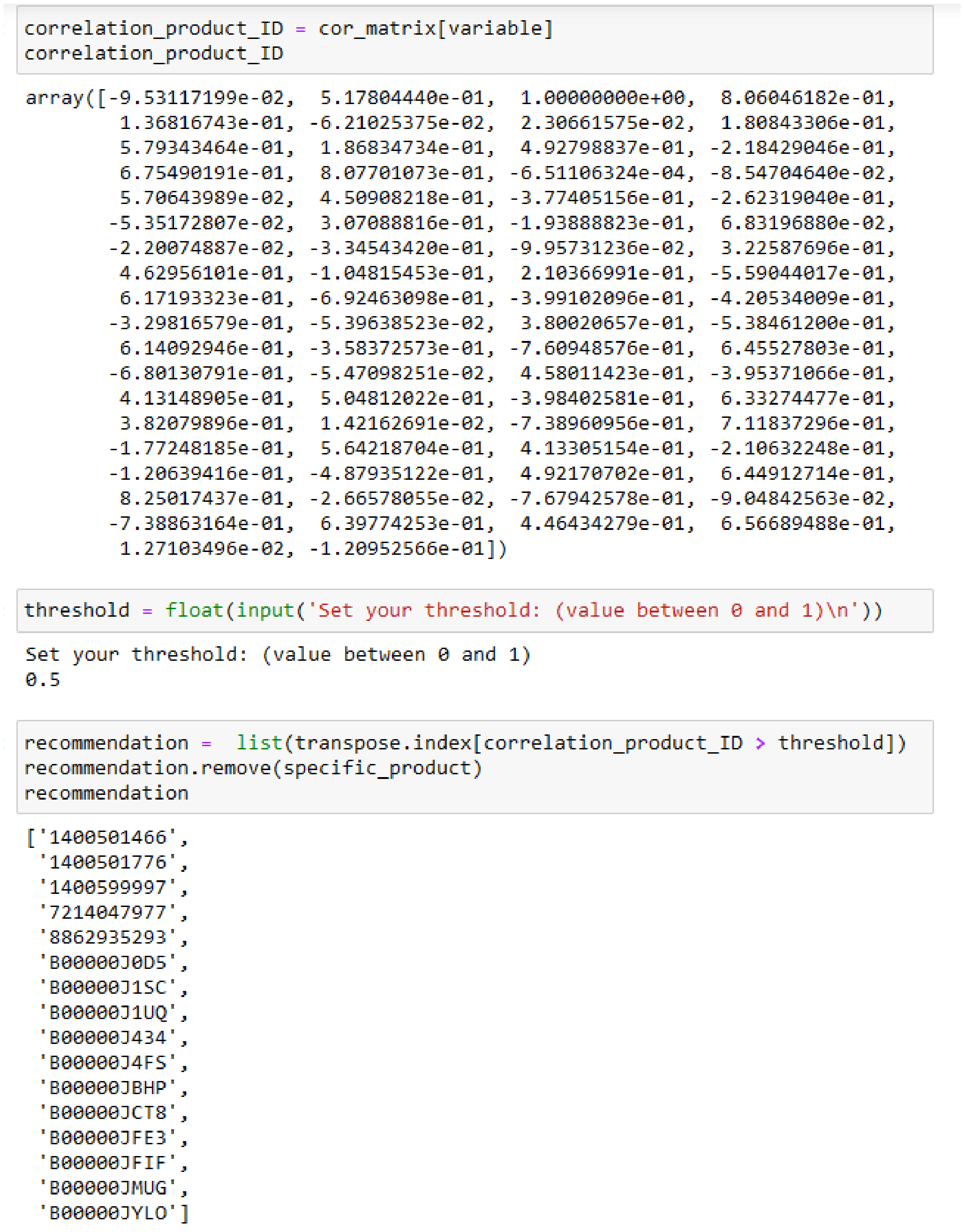
10 rows × 9832 columns

```
In [35]: from sklearn.decomposition import TruncatedSVD
         SVD = TruncatedSVD(n_components=15)
         decomposed_matrix = SVD.fit_transform(transpose)
         print(decomposed_matrix.shape)
         decomposed_matrix

         (74, 15)

Out[35]: array([[ 1.48104508e+02, -1.54623412e-01,  2.45535123e-02, ...,
                  1.45816948e-03, -2.61197391e-03, -4.70599314e-04],
                [ 6.83555797e-04,  1.28212110e+00,  1.48173836e+00, ...,
                 -2.32066161e+00, -1.03440627e-01, -7.06083007e-01],
                [ 3.80909921e-05,  9.96011887e-02,  3.14796512e-01, ...,
                  4.19662426e-02, -1.29819897e-01,  6.39639352e-01],
                ...,
                [ 1.84190639e-04,  3.14143484e-03, -1.39672439e-04, ...,
                  3.50083860e-01, -2.16285818e-01,  7.71354315e-01],
                [ 7.17675664e-06,  6.54111934e-03, -8.76337272e-04, ...,
                 -1.61531537e-03, -2.37198475e-02,  1.42182426e+00],
                [-3.14071058e-06,  4.92699928e-04,  6.39678244e-04, ...,
                 -1.17093485e-01,  1.03363346e-01,  6.90555856e-02]])
```

We then turned the decomposed matrix into a correlation matrix using *np.corrcoef()*, which returns the Pearson product-moment correlation coefficients (item-item correlation). The

Pearson product-moment correlation coefficient quantifies the linear link between two questions/measures/variables, X and Y. The correlation coefficient might range between +1 and -1. A positive correlation (e.g., +0.32) indicates that X and Y have a positive relationship.

Then all we have to do is choose an item from the list and recommend items whose correlation coefficient is above a certain positive threshold (ranging from 0 to 1). The closer the coefficient is to 1, the more similar it is.



```
correlation_product_ID = cor_matrix[variable]
correlation_product_ID
```

```
array([-9.53117199e-02,  5.17804440e-01,  1.00000000e+00,  8.06046182e-01,
        1.36816743e-01, -6.21025375e-02,  2.30661575e-02,  1.80843306e-01,
        5.79343464e-01,  1.86834734e-01,  4.92798837e-01, -2.18429046e-01,
        6.75490191e-01,  8.07701073e-01, -6.51106324e-04, -8.54704640e-02,
        5.70643989e-02,  4.50908218e-01, -3.77405156e-01, -2.62319040e-01,
       -5.35172807e-02,  3.07088816e-01, -1.93888823e-01,  6.83196880e-02,
       -2.20074887e-02, -3.34543420e-01, -9.95731236e-02,  3.22587696e-01,
        4.62956101e-01, -1.04815453e-01,  2.10366991e-01, -5.59044017e-01,
        6.17193323e-01, -6.92463098e-01, -3.99102096e-01, -4.20534009e-01,
       -3.29816579e-01, -5.39638523e-02,  3.80020657e-01, -5.38461200e-01,
        6.14092946e-01, -3.58372573e-01, -7.60948576e-01,  6.45527803e-01,
       -6.80130791e-01, -5.47098251e-02,  4.58011423e-01, -3.95371066e-01,
        4.13148905e-01,  5.04812022e-01, -3.98402581e-01,  6.33274477e-01,
        3.82079896e-01,  1.42162691e-02, -7.38960956e-01,  7.11837296e-01,
       -1.77248185e-01,  5.64218704e-01,  4.13305154e-01, -2.10632248e-01,
       -1.20639416e-01, -4.87935122e-01,  4.92170702e-01,  6.44912714e-01,
        8.25017437e-01, -2.66578055e-02, -7.67942578e-01, -9.04842563e-02,
       -7.38863164e-01,  6.39774253e-01,  4.46434279e-01,  6.56689488e-01,
        1.27103496e-02, -1.20952566e-01])
```

```
threshold = float(input('Set your threshold: (value between 0 and 1)\n'))
```

```
Set your threshold: (value between 0 and 1)
0.5
```

```
recommendation =  list(transpose.index[correlation_product_ID > threshold])
recommendation.remove(specific_product)
recommendation
```

```
['1400501466',
 '1400501776',
 '1400599997',
 '7214047977',
 '8862935293',
 'B00000J0D5',
 'B00000J1SC',
 'B00000J1UQ',
 'B00000J434',
 'B00000J4FS',
 'B00000JBHP',
 'B00000JCT8',
 'B00000JFE3',
 'B00000JFIF',
 'B00000JMUG',
 'B00000JYLO']
```

Limitations

Due to limitation of RAM, we had to drop the timestamp column, reduce the number of observations to 50,000 , and we could not implement cross validation (k-fold)

Conclusion

The three approaches had different results. The popularity-based model recommended the most popular items rated. The memory-based collaborative filtering approach used KNN with means to get similarity between users and items. And the model-based collaborative filtering used truncatedSVD to get a correlation matrix for all the items, and recommend the items based on the positive correlation between two different items.

References

https://surprise.readthedocs.io/en/stable/getting_started.html

https://scikit-learn.org/stable/

Alamdari, P. M., Navimipour, N. J., Hosseinzadeh, M., Safaei, A. A., & Darwesh, A. (2020). A systematic study on the recommender systems in the e-commerce. *IEEE Access*, *8*, 115694–115716. https://doi.org/10.1109/access.2020.3002803