Sitkar Stepan, Voitovych Oksana P., Horbatiuk Roman, Sitkar Taras, Shrol Tetiana, Poliukhovych Nataliia, Grygus Igor, Zukow Walery. The intellectual system of movies recommendations based on the collaborative filtering. Journal of Education, Health and Sport. 2022;12(3):115-127. eISSN 2391-8306. DOI http://dx.doi.org/10.12775/JEHS.2022.12.03.010
https://apcz.umk.pl/JEHS/article/view/JEHS.2022.12.03.010
https://zenodo.org/record/6362188

The journal has had 40 points in Ministry of Education and Science of Poland parametric evaluation. Annex to the announcement of the Minister of Education and Science of December 21, 2021. No. 32343.
Has a Journal's Unique Identifier: 201159. Scientific disciplines assigned: Physical Culture Sciences (Field of Medical sciences and health sciences); Health Sciences (Field of Medical Sciences and Health Sciences).

Punkty Ministerialne z 2019 - aktualny rok 40 punktów. Załącznik do komunikatu Ministra Edukacji i Nauki z dnia 21 grudnia 2021 r. L.p. 32343. Posiada Unikatowy Identyfikator Czasopisma: 201159.
Przypisane dyscypliny naukowe:Nauki o kulturze fizycznej (Dziedzina nauk medycznych i nauk o zdrowiu); Nauki o zdrowiu (Dziedzina nauk medycznych i nauk o zdrowiu).

Received: 10.01.2022. Revised: 28.02.2022. Accepted: 16.03.2022.

The intellectual system of movies recommendations based on the collaborative filtering

Stepan Sitkar
Ternopil Volodymyr Hnatiuk National Pedagogical University, Ternopil, Ukraine

Oksana P. Voitovych
Rivne State University of Humanities, Rivne, Ukraine

Roman Horbatiuk
Ternopil Volodymyr Hnatiuk National Pedagogical University, Ternopil, Ukraine

Taras Sitkar
Ternopil Volodymyr Hnatiuk National Pedagogical University, Ternopil, Ukraine

Tetiana Shrol
Rivne State University of Humanities, Rivne, Ukraine

Nataliia Poliukhovych
Rivne State University of Humanities, Rivne, Ukraine

Igor Grygus
National University of Water and Environmental Engineering, Rivne, Ukraine

Walery Zukow
Nicolaus Copernicus University, Torun, Poland
w.zukow@wp.pl

Abstract. The investigation deals with designing and developing of intellectual system of movies recommendations based on the collaborative filtering using the Python software environment. In particular, the approaches (Content-based approach, Collaborative filtering, Hybrid models) in recommendatory system construction with the help of neural networks have been analyzed. It has been established that it is difficult to implement and learn the Content-based approach and it strongly depends on

the subject area. Collaborative filtering is more simple in implementation, training, it is universal, but it has a flaw in the form of a «cold-start». Accordingly, the collaborative filtering has been chosen for the design and development of the intellectual system of movies recommendations. While designing a system of recommendations based on collaborative filtering, the Naive Recommendations, Recommendations based on average ratings of similar users, Recommendations based on average user ratings and similarity matrix have been described; their algorithm and their implementation using the Python software environment have been demonstrated. As a result the intellectual system of recommendations has been realized and it can offer a movie to the user according to his/her preferences.

Key words: Collaborating filtering, recommendation system, neural network, naive recommendation, recommendations based on average ratings of similar users, recommendations based on average user ratings and similarity matrix.

1 Introduction

Today, there is no doubt that the recommendation systems occupy a leading positions in the development process of artificial intelligence. We can assert that the use of recommendation systems has entered in everyday life and it comes as no surprise to us. Nowadays, such systems have been used in many web resources. However, not so many people are aware of the principles of functioning and algorithm creation of such systems. The implementation of such systems will help to understand the principles of work and will provide the opportunities for the improvement of existing algorithms. In this study, the creation features of movies recommendation systems based on the collaborative filtering has been revealed, it allows to propose movies in accordance with users' aesthetic preferences.

2 Problem statement

The main task of the investigation is the construction of movies recommendations system based on the collaborative filtering using the Python software environment. In the process of the main task solving, a number of secondary tasks have been solved, in particular: the general structure of the recommendations system has been examined; the analysis and characteristic of the main approaches in recommendation systems implementation; the particular use of collaborative filtering in the recommendation systems has been established; a description of positive and negative sides of models in the collaborative filtering has been presented. In the process of tasks solving, the movies recommendations system based on collaborative filtering will be created, and the system training, the checking correctness of the recommendations data, the quantitative and qualitative obtaining of system characteristics will be conducted.

3 The construction of the movies recommendations system based on the models of collaborative filtering

Nowadays, there are three approaches to constructing the recommendation systems (Fig. 1)

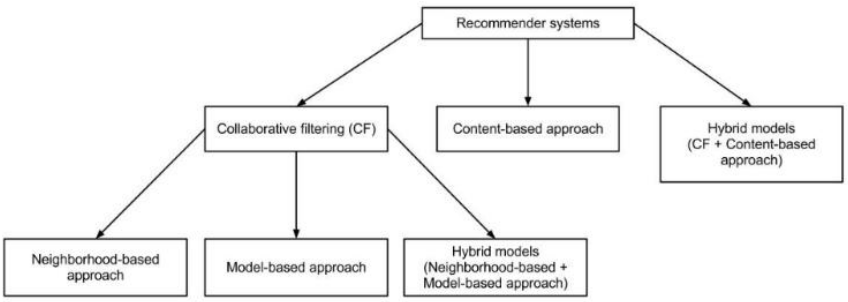
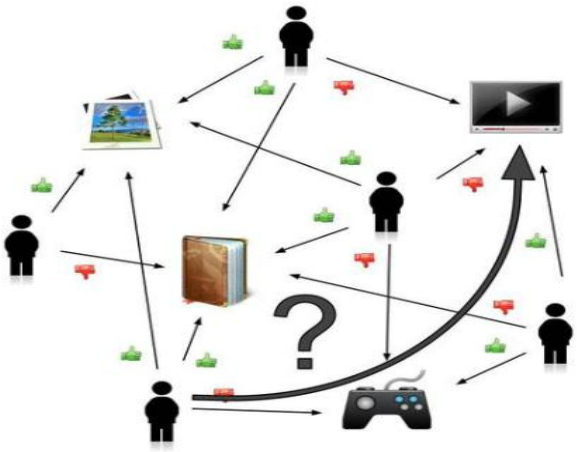


Fig.1 Approaches to constructing recommendation systems

Content-based approach. This approach is based on the objects description which we should recommend (recommend movies that are similar to those that a user liked). The realization of this approach is rather complicated. And the complexity is that it is necessary to take into account the large number of characteristics variety of the objects which are recommended and it is tightly connected with a specific subject area. Therefore, the use of this approach requires a large power of computing machines, and the results are commensurate with other approaches[9,10,11].
Collaborative filtering. This approach is based on the user rating and its similarity to other users (Fig. 2). This approach is much more simple in implementation, it does not require too powerful computational capabilities. However, it has disadvantage and it is associated with the so-called «cold start», so what we should recommend the user who has not chosen anything yet [2; 4; 5].

Fig.2 Collaborative filtering



Hybrid models. This approach combines the previous two approaches. It has the best productivity, but it also has disadvantages of the previous two approaches. Taking into account the information mentioned above it is decided to design and implement a system of recommendations based on collaborative filtering [6,7,8,9].
For system teaching, was used a dataset [1] with users' ratings of different movies. The dataset was quite voluminous (671 unique users, 9066 original movies and more than 1 million reviews). For teaching system, 100000 entries from the entire set (n = 100000) will be enough (Table 1).

```
import numpy as np
import pandas as pd
ratings_df = pd.read_csv('./ml-20m/ratings.csv')
print('Unique users count:
{}'.format(len(ratings_df['userId'].unique())))
```

```
print('Unique movies count:
{}'.format(len(ratings_df['movieId'].unique())))
print('DataFrame shape: {}'.format(ratings_df.shape))
ratings_df.head()
# Unique users count: 671
# Unique movies count: 9066
# DataFrame shape: (100004, 4)
n = 100000
ratings_df_sample = ratings_df[:n]
n_users = len(ratings_df_sample['userId'].unique())
n_movies = len(ratings_df_sample['movieId'].unique())
(n_users, n_movies)
# (671, 9066)
```

Table 1. DataFrame example

	userId	***movieId***	***rating***	***timestamp***
0	1	1	2.5	1260759144
1	1	2	3.0	1260759179
2	1	3	3.0	1260759182
3	1	4	2.0	1260759185
4	1	5	4.0	1260759205

For a comfortable information processing, we format our set of movies IDs in such a way: they begin with 1 and end with *n_movies*

```
movie_ids = ratings_df_sample['movieId'].unique()
def scale_movie_id(movie_id):
    scaled = np.where(movie_ids == movie_id)[0][0] + 1
    return scaled
ratings_df_sample['movieId'] =
ratings_df_sample['movieId'].apply(scale_movie_id)
ratings_df_sample.head()
```

Divide the all dataset into two parts: a teaching set and a test set. It is clear that the first will be used for teaching, and the second one will measure the quality of the predicted ratings. Divide the set with the help of *train_test_split* function from the scikit-learn module:

```
train_test_split з модуля scikit-learn:
from sklearn import cross_validation as cv
train_data, test_data = cv.train_test_split(ratings_df_sample,
test_size=0.2)
print('Train shape: {}'.format(train_data.shape))
print('Test shape: {}'.format(test_data.shape))
# Train shape: (80000, 4)
# Test shape: (20000, 4)
```

To determine the quality of the predicted ratings, we use the measure of RMSE (Root Mean Square Error,) (1)[18,19,20]:

$$RMSE = \sqrt{\frac{1}{|D|}\sum_{(u,i\,\in D)} (\widehat{r_{u,i}} - r_{u,i})^2} \qquad (1)$$

The root mean square error is the root of the average error of all ratings given by our algorithm. For example, let us assume, we have the following set of ratings (Table 2):

Table 2. Example the set of ratings

user	***movie***	***rating***	***predicted***
Anna	Thor 1	5	3
Anna	Thor 2	5	5
Anna	Thor 3	5	1
Ben	Thor 1	4	4
Ben	Thor 2	1	3
Ben	Ant-man	5	3
Ben	Hulk	3	1
Lora	Thor 1	1	5
Lora	Ant-man	5	5
Sam	Thor 1	5	5
Sam	Thor 3	5	5
Sam	Hulk	5	1

The predicted column contains the predicted rating algorithms (in fact, as well as ratings in the rating column). In this case, the RMSE will be (2):

$$RMSE = \sqrt{5.333} = 2.309 \quad (2)$$

```
from sklearn.metrics import mean_squared_error
from math import sqrt
def rmse(prediction, ground_truth):
    prediction =
np.nan_to_num(prediction)[ground_truth.nonzero()].flatten()
    ground_truth =
np.nan_to_num(ground_truth)[ground_truth.nonzero()].flatten()
    mse = mean_squared_error(prediction, ground_truth)
    return sqrt(mse)
```

Let's create matrixes of size (n_users, n_movies) for the teaching and test sets in such a way that the element in the cell [i, j] reflects the rating of the i-user of the j-movie.

```
train_data_matrix = np.zeros((n_users, n_movies))
for line in train_data.itertuples():
    train_data_matrix[line[1] - 1, line[2] - 1] = line[3]
    test_data_matrix = np.zeros((n_users, n_movies))
for line in test_data.itertuples():
    test_data_matrix[line[1] - 1, line[2] - 1] = line[3]
```

It should be noted, that some of the rating got into the teaching set of train_data_matrix, and the other part is – test_data_matrix, so that you can measure the quality of the predictions.

```
from  sklearn.metrics.pairwise import pairwise_distances
user_similarity = pairwise_distances(train_data_matrix,
metric='cosine')
item_similarity = pairwise_distances(train_data_matrix.T,
metric='cosine')
```

One of the important moments in collaborative filtering is to find the similar users for User-Based and the similar objects (in our case, movies) for Item-Based Collaborative Filtering. There are different approaches to this. One of them is to use the cosine distance between vectors which describe the users and objects. There is a predefined function in the scikit-learn module.

That is to say, the user_similarity [i] [j] is the cosine distance between the i- line and the j-line (you can check through spicy.spatial.distance cosine (x,y)) and item similarity [i][j]] is the cosine distance between the i- and j- columns.

It can be assumed that the cosine distance represents the degree of similarity. The more users or movies similar to each other - the smaller cosine distance will be.

```
from scipy.spatial import distance
print(distance.cosine([2,2],[1,1]))
print(distance.cosine([3,3],[2,3]))
print(distance.cosine([3, 3],[1, 1.5]))
print(distance.cosine([3, 3],[1, 3]))
# 2.2204460492503131e-16
# 0.019419324309079666
# 0.019419324309079666
# 0.10557280900008403
```

We use two methods: one for User-base collaborative filtering (based on the similarity of users) and the second for Item-Based Collaborative Filtering (based on the similarity of movies). In fact, this is one and the same algorithm, but in the second case, a transposed matrix of ratings is used (movies are arranged in lines, and the users are in columns).

After finishing the preparatory phase, we will proceed directly to the implementation of collaborative filtering models. We use the following three models for the recommendation system of collaborative filtering: Naive Recommendation, Recommendations based on average ratings of similar users, Recommendations based on average user ratings and similarity matrix.

Naive Recommendation. The simplest recommendation system which calculates the predicted ratings of the user u of the movie i by the formula (3):

$$r_{u,i} = \frac{\sum_{u' \in U} r_{u',i}}{N} \qquad (3)$$

where N – number of users, similar to user u; U is a set of N similar users, u' is a user similar to u (from plural U), $r_{u',i}$ – user rating u' of the movie i, $r_{u,i}$ – the rating of the movie i is provided [11,12,13].

According to formula (3), the rating of the movie i and the user u is equal to the average rating of the movie i from N users most similar to the user u.

Let's observe an example (we'll go back to it from now on). We have a set of user ratings of the following form (Table 3):

Table 3. Set of user ratings

user	***movie***	***rating***
Anna	Thor 1	5
Anna	Thor 2	5
Anna	Thor 3	5
Ben	Thor 1	4
Ben	Thor 2	1
Ben	Ant-man	5
Ben	Hulk	3
Lora	Thor 1	1
Lora	Ant-man	5
Sam	Thor 1	5
Sam	Thor 3	5
Sam	Hulk	5

For more convenient work, this table is presented in another type (Table 4):

Table 4. Another type of data representation

	Thor 1	***Thor 2***	***Thor 3***	***Ant-man***	***Hulk***
Anna	5	5	5	0	0
Ben	4	1	0	5	3
Lora	1	0	0	5	0
Sam	5	0	5	0	4

It has been noted before, we will use a cosine distance to determine the proximity. For convenience, we will rewrite the matrix in such a way that the lines and columns have the user names (Table 5):

```
from  sklearn.metrics.pairwise import pairwise_distances
demo_data = [[5,5,5,0,0], [4,1,0,5,3], [1,0,0,5,0],
[5,0,5,0,4]]
pairwise_distances(demo_data, metric='cosine')
# array([[ 0., 0.59577396, 0.8867723, 0.28933095],
#        [ 0.59577396, 0., 0.2036092, 0.4484398 ],
#        [ 0.8867723 , 0.2036092, 0., 0.87929886],
#        [ 0.28933095, 0.4484398, 0.87929886,  0.        ]])
```

Table 5. Cosine distance to determine the proximity

	Anna	***Ben***	***Lora***	***Sam***
Anna	0	0.5957	0.8867	0.2893
Ben	0.5957	0	0.2036	0.4484
Lora	0.8867	0.2036	0	0.8792
Sam	0.2893	0.4484	0.8792	0

On the basis of matrix resulting, the number in the cell [i,j] reflects the similarity of users *i* and *j*. In this example, the number 0.59577396 in the cell [0,1] is the cosine distance between the rating of Anna and Ben.

Let's assume that *N* is equal to two. The two persons who will be the most similar to Anna users will be Sam (distance is 0.2893) and Ben (distance is one of 0.5957); for Ben - Lora and Sam (distances of 0.2036 and 0.4484 accordingly); for Lora - Ben and Sam (distance 0.2036 and 0.8792 accordingly); for Sam – Anna and Ben (distance 0.2893 and 0.4484accordingly).

Using formula (3) we calculate Anna's predicted rating for Ant-man and Hulk movies.

For Ant-man: $$r_{Anna,Ant-man} = \frac{5+0}{2} = 2{,}5 \quad (4)$$

For Hulk: $$r_{Anna,Hulk} = \frac{3+4}{2} = 3{,}5 \quad (5)$$

```
# User-based collaborative filtering
def naive_predict(top):
    top_similar_ratings = np.zeros((n_users, top, n_movies))
    for i in range(n_users):
        top_sim_users = user_similarity[i].argsort()[1:top + 1]
        top_similar_ratings[i] =
train_data_matrix[top_sim_users]
    pred = np.zeros((n_users, n_movies))
    for i in range(n_users):
        pred[i] = top_similar_ratings[i].sum(axis=0)/top
    return pred
def naive_predict_item(top):
    top_similar_ratings = np.zeros((n_movies, top, n_users))
    for i in range(n_movies):
        top_sim_movies = item_similarity[i].argsort()[1:top +
1]
        top_similar_ratings[i] =
train_data_matrix.T[top_sim_movies]
    pred = np.zeros((n_movies, n_users))
    for i in range(n_movies):
        pred[i] = top_similar_ratings[i].sum(axis=0)/top
    return pred.T
naive_pred = naive_predict(7)
print('User-based CF RMSE: ', rmse(naive_pred,
test_data_matrix))
naive_pred_item = naive_predict_item(7)
print('Item-based CF RMSE: ', rmse(naive_pred_item,
test_data_matrix))
# User-based CF RMSE:  2.81961691384066
# Item-based CF RMSE:  3.001291898703705
```

Recommendations based on average ratings of similar users

The more complicated implementation requires the use of matrix of the similarity and ratings of «similar» users. The formula for calculating the predicted ratings (ratings) (6):

$$r_{u,i} = \frac{\sum_{u' \in U} simil(u,u') r_{u',i}}{\sum_{u' \in U} |simil(u,u')|} \quad (6)$$

where - *simil(u,u')* - the «similarity» of the user *u* and user *u'*, - $r_{u',i}$ - the user's rating *u* from the *U* movie *i*, - *u, u'*- are similar to the previous formula.

Thus, the rating that was predicted for a movie will be equal to the sum of the products of the «similarity» of the user to his/her rating in all the most similar users [14,15].

We will calculate predicted Anne's ratings for Ant-man and Hulk by the formula (6).

$$\text{For Ant-man: } r_{Anna,Ant-man} = \frac{0{,}5957*5+0{,}2893*0}{|0{,}5957|+|0{,}2893|} = 3{,}365537 \qquad (7)$$

$$\text{For Hulk: } r_{Anna,Hulk} = \frac{0{,}5957*3+0{,}2893*5}{|0{,}5957|+|0{,}2893|} = 3{,}6537 \qquad (8)$$

```
def k_fract_predict(top):
    top_similar = np.zeros((n_users, top))
    for i in range(n_users):
        user_sim = user_similarity[i]
        top_sim_users=user_sim.argsort()[1:top+1]#[-top:]
        for j in range(top):
            top_similar[i, j] = top_sim_users[j]
    abs_sim = np.abs(user_similarity)
    pred = np.zeros((n_users, n_movies))
    for i in range(n_users):
        indexes = top_similar[i].astype(np.int)
        numerator = user_similarity[i][indexes]
        product = numerator.dot(train_data_matrix[indexes])
        denominator =
abs_sim[i][top_similar[i].astype(np.int)].sum()
        pred[i] = product / denominator
    return pred
def k_fract_predict_item(top):
    flag = True
    top_similar = np.zeros((n_movies, top))
    for i in range(n_movies):
        movies_sim = item_similarity[i]
        top_sim_movies = movies_sim.argsort()[1:top + 1]
        for j in range(top):
            top_similar[i, j] = top_sim_movies.T[j]
    abs_sim = np.abs(item_similarity)
    pred = np.zeros((n_movies, n_users))
    for i in range(n_users):
        indexes = top_similar[i].astype(np.int)
        numerator = item_similarity[i][indexes]
        product = numerator.dot(train_data_matrix.T[indexes])
        denominator = abs_sim[i][indexes].sum()
        denominator=denominator if denominator!=0 else 1
        pred[i] = product/denominator
    return pred.T
k_predict = k_fract_predict(7)
print('User-based CF RMSE: ', rmse(k_predict,
test_data_matrix))
k_predict_item = k_fract_predict_item(7)
```

```
print('Item-based CF RMSE: ', rmse(k_predict_item,
test_data_matrix))
# User-based CF RMSE:  2.821055763616836
# Item-based CF RMSE:  3.245023071118644
```

As a result of realization with the same input data, the result turned out to be a little worse than at the first implementation.

Recommendations based on the average user ratings and similarity matrix.

This implementation depends on the ratings that the user set before (to be more precisely, from the average rating for all movies that were rated by the users), average ratings of «similar» users, and similarity coefficients:

$$r_{u,i} = \bar{r_u} + \frac{\sum_{u' \in U} simil(u,u')(r_{u',i} - \bar{r_{u'}})}{\sum_{u' \in U} |simil(u,u')|} \quad (9)$$

where $\bar{r}$ - the average rating of the user, and the other formula variables have been already discussed above.[16,17]

We use formula (9) to predict Anna's movies Ant-man and Hulk. For this, the average ratings for all rated movies for Anna, Sam and Ben will be required. They are equal to 5, 4.666667, 3.25 accordingly.

The rate for Ant-man movie:

$$r_{Anna,Ant-man} = 5 + \frac{0{,}5957 * (3{,}25 - 5) + 0{,}2893 * (4{,}666 - 0)}{|0{,}5957| + |0{,}2893|} = 5{,}414 \quad (10)$$

The rate for Hulk movie:

$$r_{Anna,Hulk} = 5 + \frac{0{,}5957*(3{,}25-5)+0{,}2893*(4{,}666-0)}{|0{,}5957|+|0{,}2893|} = \quad = 5{,}414 \quad (11)$$

In the process of realization of the given model, the obtained data were more than 5, which is not possible. However, this result we interpreted as an unsuccessfully chosen set for an example. If we make a calculation for the whole dataset, then we will get the correct results.

```
def k_fract_mean_predict(top):
    top_similar = np.zeros((n_users, top))
    for i in range(n_users):
        user_sim = user_similarity[i]
        top_sim_users = user_sim.argsort()[1:top + 1]
        for j in range(top):
            top_similar[i, j] = top_sim_users[j]
    abs_sim = np.abs(user_similarity)
    pred = np.zeros((n_users, n_movies))
    for i in range(n_users):
        indexes = top_similar[i].astype(np.int)
        numerator = user_similarity[i][indexes]
        mean_rating = np.array([x for x in train_data_matrix[i]
if x>0]).mean()
        diff_ratings = train_data_matrix[indexes] -
train_data_matrix[indexes].mean()
        numerator = numerator.dot(diff_ratings)
```

```
        denominator =
abs_sim[i][top_similar[i].astype(np.int)].sum()
        pred[i] = mean_rating + numerator/denominator
    return pred
def k_fract_mean_predict_item(top):
    top_similar = np.zeros((n_movies, top))
    for i in range(n_movies):
        movie_sim = item_similarity[i]
        top_sim_movies = movie_sim.argsort()[1:top+1]
        for j in range(top):
            top_similar[i, j] = top_sim_movies[j]
    abs_sim = np.abs(item_similarity)
    pred = np.zeros((n_movies, n_users))
    for i in range(n_movies):
        indexes = top_similar[i].astype(np.int)
        numerator = item_similarity[i][indexes]
        mean_rating = np.array([x for x in
train_data_matrix.T[i] if x>0]).mean()
        mean_rating = 0 if np.isnan(mean_rating) else
mean_rating
        diff_ratings = train_data_matrix.T[indexes]-
mean_rating
        numerator = numerator.dot(diff_ratings)
        denominator=
abs_sim[i][top_similar[i].astype(np.int)].sum()
        denominator=denominator if denominator!=0 else 1
        pred[i] = mean_rating+numerator/denominator
    return pred.T
k_predict = k_fract_mean_predict(7)
print('User-based CF RMSE: ', rmse(k_predict,
test_data_matrix))
k_predict_item = k_fract_mean_predict_item(7)
print('Item-based CF RMSE: ', rmse(k_predict_item,
test_data_matrix))
# User-based CF RMSE: 1.5491818781971805
# Item-based CS RMSE: 3.1094062597156267
```

In this option we has got the best result for *User-based Collaborative Filtering*. For Item-based, the best one is the first implementation.

4 Conclusion

The investigation confirmed that the best for User-Based Collaborative Filtering in our dataset has been the third approach, and for Item-Based is the first one.

Since the cosine distance between the two vectors was considered, then the best result will be the one where we get the least value. The summary of the obtained results is presented in the table 6:

Table 6. The summary of the obtained

	Method 1	Method 2	Method 3
User-Based	2.819	2.821	1.549
Item-Based	3.001	3.245	3.109

As we can see from the table, User-Based gives twice better results compared to Item-Based. This is due to the fact that it is easier to find a similar user than all similar movies which were rated by the user.

In the future, the created system of movies recommendations based on the collaborative filtering can be expanded and added Content-based approach and Hybrid models, which will improve the quality of this system.

References

1. Films data set, https://grouplens.org/datasets/movielens
2. Derrick Mwiti, “How to build a Simple Recommender System in Python” - https://towardsdatascience.com/how-to-build-a-simple-recommender-system-in-python-375093c3fb7d
3. Collaborative filtering (Wikipedia) -https://en.wikipedia.org/wiki/Collaborative_filtering
4. Chhavi Saluja, “Collaborative Filtering based Recommendation Systems exemplified.” - https://towardsdatascience.com/collaborative-filtering-based-recommendation-systems-exemplified-ecbffe1c20b1
5. Steeve Huang, “Introduction to Recommender System. Part 1 (Collaborative Filtering, Singular Value Decomposition)” - https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value-decomposition-44c9659c5e75
6. D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. Communications of the ACM, 35(12):61--70, 1992. Google ScholarDigital Library
7. J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In Proceedings of the 1999 Conference on Research and Development in Information Retrieval, Aug. 1999. Google ScholarDigital Library
8. J. Kleinberg and D. Liben-Nowell. The link prediction problem for social networks. In Proceedings of the 12th International Conference on Information and Knowledge Management (CIKM), 2003. Google ScholarDigital Library
9. J. M. Pujol, R. Sanguesa, and J. Delgado. Extracting reputation in multi agent systems by means of social network topology. In Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2002. Google ScholarDigital Library
10. M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining (KDD), 2002. Google ScholarDigital Library
11. Hill, W. et al. Recommending and Evaluating Choices in a Virtual Community of Use. In ”. In Proceedings of ACM CHI’95 Conference on Human Factors in Computing Systems, pages 194–201. 1995.
12. Resnick, P., et al. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In Proceedings of ACM CSCW’94 Conference on Computer-Supported Cooperative Work, pages 175–186. 1994.

13. Fleder D., Hosanagar K. Blockbuster Culture's Next Rise or Fall: The Impact of Recommender Systems on Sales Diversity (англ.) // Management Science, Vol. 55, No. 5, May 2009, pp. 697–712. P. 1–49.
14. Xiaoyuan Su and Taghi M. Khoshgoftaar. A Survey of Collaborative Filtering Techniques A Survey of Collaborative Filtering Techniques. Hindawi Publishing Corporation, Advances in Artificial Intelligence archive, USA : журнал. 2009. P. 1–19.
15. Yehuda Koren. Factor in the Neighbors: Scalable and Accurate Collaborative Filtering (англ.) // Yahoo! Research, Haifa. P. 1–11.
16. Linden G., Smith B., and York J. Item-to-Item Collaborative Filtering. IEEE Internet Computing, Los Alamitos, CA USA. 2003. P. 76–80.
17. Sarwar B., Karypis G., Konstan J., and Riedl J. Item-Based Collaborative Filtering Recommendation Algorithms. University of Minnesota, Minneapolis. WWW10, Hong Kong, May 1–5, 2001. 2001. P. 285–295.
18. Melville P., Mooney R., Nagarajan R. Content-Boosted Collaborative Filtering for Improved Recommendations. University of Texas, USA : AAAI-02, Austin, TX, USA, 2002. 2002. P. 187–192.
19. Zan Huang, Xin Li, Hsinchun Chen. Link Prediction Approach to Collaborative Filtering. University of Arizona, USA : JCDL'05, Denver, Colorado, USA, June 7–11, 2005.
20. Sammut C., Webb J. (Eds.). Encyclopedia of Machine Learning. NY, USA: IBM T. J. Watson Research Center, 2010. T. 1. P. 829–838. 1031 c. ISBN 978-0-387-30768-8.