Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 176 (2020) 1940–1949

Procedia Computer Science

www.elsevier.com/locate/procedia

24th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Using Differential Evolution in order to create a personalized list of recommended items

Urszula Boryczka, Michał Bałchanowski*

Institute of Computer Science, University of Silesia in Katowice, ul. Bedzińska 39, Sosnowiec 41-200, Poland

Abstract

The recommendation systems are used to suggest new, still not discovered items to users. At the moment, in order to achieve the best quality of the generated recommendations, users and their choices in the system must be analyzed to create a certain profile of preferences for a given user in order to adjust the generated recommendation to his personal taste. This article will present a recommendation system, which based on the Differential Evolution (DE) algorithm will learn the ranking function while directly optimizing the average precision (AP) for the selected user in the system. To achieve that, items are represented through a feature vectors generated using user-item matrix factorization. The experiments have been conducted on a popular and widely available public dataset MovieLens, and show that our approach in certain situations can significantly improve the quality of the generated recommendations. Results of experiments are compared with other techniques.

Keywords: recommendation systems; learning to rank; ranking function; differential evolution; matrix factorization

1. Introduction

The progressive computerization of the world makes more and more products, services and information available on the Internet. Additionally, wide access to the Internet and the development of mobile technology allows almost every user to be constantly ”online” and look for advice of the Web in making everyday decisions. Unfortunately, there is so much information available that an average user is often unable to familiarize oneself with it and choose the best option for himself. This phenomenon is known as ”information overload”. The issue has been partially solved by the means of search engines, which try to match the returned results with the query provided by the user. Unfortunately, the disadvantage of this approach is that the user must know how to construct the query in the correct way. This is where the Recommendation Systems come in, systems which do not require the user to make such a query, but offer

* Corresponding author.
E-mail address: urszula.boryczka@us.edu.pl, michal.balchanowski@us.edu.pl

1877-0509
10.1016/j.procs.2020.09.233

the user certain items based on his previous interactions with other items in the system. Owing to this approach, the user can discover new items that he would not even think about when formulating a search engine query.

The problem of recommendations can be presented as a problem of finding such a ordering of the content suggested to the user that the resources that are relevant are displayed at the top of the recommended list. This problem is not trivial [9] and heuristic algorithms are often used to solve it, due to the fact that the measures used to measure the quality of the generated recommendation are difficult to optimize directly using standard optimization algorithms.

In order to make it possible for the system to match preferences for a particular user, the resources in the system must have certain features. Unfortunately, it often happens that we do not have access to these features or these features are of a very poor quality. In order to solve this problem, a matrix factorization technique [17] has been proposed, owing to which we will obtain certain features describing items in the system that cannot be directly observed, referred to as "latent features".

The main contribution of this paper is to present the use of the Differential Evolution (DE) algorithm to directly optimize the average precision (AP) measure in recommendation systems. Although there are papers describing the use of DE for this purpose in information retrieval systems [6], but to the best of our knowledge there is no work describing usage of DE for this kind of task in recommendation systems. In addition, most of the work in the recommendation systems field mainly focuses on predicting the ratings that the user would give to specific items, rather than creating a list of recommended items, which is certainly closer to the task of recommendation.

The article is divided as follows: the next chapter describes the literature overview. In chapter 3 the basic information about recommendation systems, matrix factorization, learning to rank and a short description of the Differential Evolution algorithm (DE) will be presented. Chapter 4 is focused on description of the system created, formulation of the fitness function and application of the DE algorithm in learning to rank. In chapter 5 will be explained how experiments were conducted. The last two chapters (6 and 7) are dedicated to the results of experiments and conclusions.

2. Literature overview

Learning to rank is very popular topic and many articles addressing this issue have been written, especially within the context of information retrieval systems. One of the more popular techniques are RankBoost [12] and RankNet [8]. It is also possible to find articles in which a metaheuristic algorithm was used to create a ranking function. Examples include Differential Evolution (DE) [6] or Particle Swarm Optimization (PSO) [11] algorithm, which, owing to its properties, can directly optimize such a measure as the mean average precision (MAP).

An excellent review of Evolutionary Algorithms in recommendation systems is paper [15] in which can be found different approaches that have been used in recent research on this topic. One of them is function learning approaches. Main example of such approach is framework called GUARD [13] which generate ranking functions by taking into account multiple objectives such as accuracy, novelty and diversity measures. Other approach based on learning a ranking function are [4] where task was to recommend relevant tags to a target object. In their next work [2] authors proposed method based on random forests and later extends experiments by other tag recommendation methods and evaluation metrics in [3].

In the recommendation systems, however, owing to the "Netflix Prize" [5] contest, among others, the most popular technique is the prediction of ratings given by the user to a given item (product). This approach, however, does not take into account how in such systems recommendations are presented to a given user. Most often recommendations are presented in the form of a list of certain items, therefore it is also important to evaluate the recommendation system in this respect [16]. In [10] the Particle Swarm Optimization (PSO) algorithm is used, which, in a similar way as described in this article, optimizes the ranking function, but the presented results, although promising, are not very extensive. It is worth noting that neural networks have also been successfully used in this problem, as described in article [20].

3. Background of the research

In this chapter we will briefly describe what a recommendation system is and how the factorization of the matrix is used to generate recommendations. Additionally, the problem of learning to rank in information retrieval systems and the Differential Evolution algorithm will be presented here.

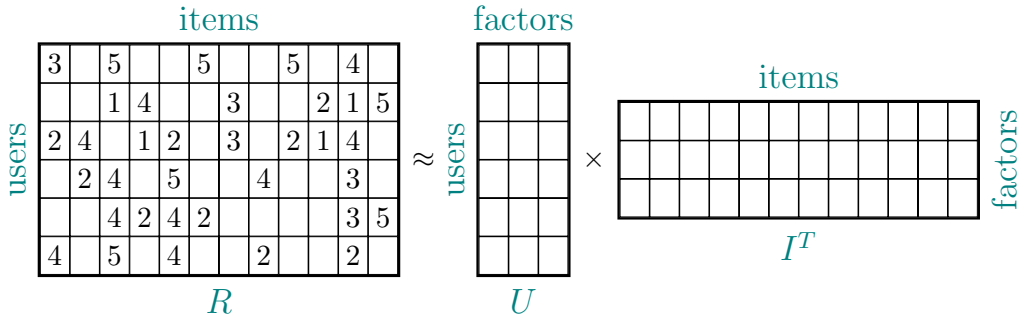

Fig. 1. Example of matrix factorization

3.1. Recommendation system

Let us consider a set of users $U = u_1, \ldots, u_{|U|}$ and a set of items $I = i_1, \ldots, i_{|I|}$. Each user $u \in U$, rated certain items $i \in I$, which will be marked r_{ui}. Therefore, the data can be presented as three (u, i, r_{ui}), which in case of our research will mean that a certain user u has rated a certain video i, by rating it r_{ui}. All ratings given by all users for all items are often presented in the form of a matrix of $m \times n$. One of the most popular data filtering techniques found in recommendation systems is the collaborative filtering technique [19]. This technique is based on the assumption that in order to recommend to a certain active user u_A new items that he has not yet rated in the system, other users are searched for who have rated items in a similar way as the u_A user and from this group of users recommendations for the u_A user are generated.

3.2. Matrix decomposition

The matrix factorization algorithm works by decomposing the user-item matrix into two smaller matrices. Owing to this solution users and items are represented by a small number of latent features. This technique became very popular in recommendation systems owing to Simon Funk, who used it in the Netflix competition, where it achieved good results [17]. Although the factorization technique for matrices in recommendation systems is similar to the popular Singular Value Decomposition (SVD) technique, it should be noted that in order to use a standard SVD on a matrix, the matrix must be complete (without empty values), which is obviously not the case in recommendation systems. For this reason factorization is done by using certain algorithms that generate the best approximation of the original matrix. In its standard form, the factorization of the matrix brings the matrix $\hat{R}$ in size $|U| \times |I|$ by the product of two smaller matrices $|U| : |U| \times k$ and $I : |I| \times k$

$$\hat{R} := UI^T \tag{1}$$

where k determines the rank of the approximated matrices. Each row of U_u in the matrix U represents the characteristics (feature vector) describing user u, and each row of I_i in the I matrix represents the characteristics (feature vector) describing item i. The formula that predicts the rating that user u has given to the item i can be described as follows:

$$\hat{r}_{ui} = U_u \cdot I_i^T \tag{2}$$

3.3. The problem of learning to rank in information retrieval systems

The problem of learning to rank for information retrieval systems is described in detail in work [18]. To put it simply, it can be said that it consists of two basic phases: training and testing. In the training phase we provide the learning algorithm with a set of queries, where each query corresponds to a set of documents. Additionally, to each document is assigned a label (e.g. human-defined), which determines its relevance to a given query. The aim of the training phase is to construct a model that will be able to predict new rankings of documents on the basis of the

queries provided in the best way. The training phase is followed by a test phase, which aims to check the quality of the generated model. In order to measure the quality between the ranking generated by the algorithm and the one in the test set, average precision (AP) metric will be used. Precision of rank n ($P@n$) measures the relevance of the first n ($TopN$) items at the top of the ranking list for a specific user and is determined as follows:

$$P@n = \frac{Number\ of\ relevant\ items\ in\ top\ n\ results}{n} \tag{3}$$

Average precision (AP) averages the $P@n$ values for different n values:

$$AP = \frac{\sum_{n=1}^{N}(P@n \times rel(n))}{Number\ of\ relevant\ items\ for\ this\ query} \tag{4}$$

where $rel(n)$ is a binary function assigning the value 1 if the nth document is relevant to the query (otherwise 0) and N is the number of items obtained. Mean average precision (MAP) averages the AP values for all U users in the system.

In addition, the problem of learning to rank can be formally written down in the following way: Let us assume a set of queries $Q = q_1, \ldots, q_{|Q|}$ and a set of documents $D = d_1, \ldots, d_{|D|}$. The training set is created from a set of query-document pairs $(q_i, d_j) \in Q \times D$, where each element is assigned a relevance score (e.g. a label) $q = (q_i, d_j)$ meaning the relationship between q_i and d_j. The query-document pair (q, d) is represented by the features vector $\phi(q, d)$. The ranking model can be written using the function:

$$f(q, d) = w^T \phi(q, d) \tag{5}$$

where w denotes a weight vector which determine the degree of significance of each feature towards the ranking score. In order to create a ranking for a given query q_i, we calculate $f(q_i, d_j)$ for each document d_j using formula 5 and sort these documents by ranking score in a descending order.

3.4. Differential Evolution

The differential evolution was developed by K. Price and R. Storn in the mid-1990s [21]. It is an evolutionary technique that has found application primarily in numerical optimization. There is a certain population of individuals who are n dimensional vectors of real numbers. As in other algorithms of this type (e.g. GA, PSO), it is based on iterative improvement of individuals in the population until a certain stop criterion is reached. Each individual is a solution to an optimization problem, and a function that evaluates the quality of an individual is called an evaluation function. Here we distinguish the following two operators: mutations and crossover [7]. The mutation is the primary operator here and can be carried out in many ways. In the basic version of the algorithm, the creation of a new individual takes place by combining three randomly selected individuals from the S population and is expressed by the following formula:

$$v_i = x_{r1} + F(x_{r2} - x_{r3}), \tag{6}$$

where $r1, r2, r3$ are three random individuals from the S population whereby $r1 \neq r2 \neq r3$. The F parameter is the amplification factor and takes a value between $0 \leq F \leq 1$. The next step is to use a crossover operator which creates a new individual u_i by combining the genotypes of parent x_i in the population S, and the individual resulting from the use of a mutation operator v_i in the population V. There is also the CR parameter, which determines the crossover probability and the function $Rand(j)$, which generates a random number in the range $(0, 1)$. The crossover process can be expressed according to the following formula:

$$u_{i,j} = \begin{cases} v_{i,j} & \text{if } (rand(j) \leq CR) \\ x_{i,j} & otherwise \end{cases} \tag{7}$$

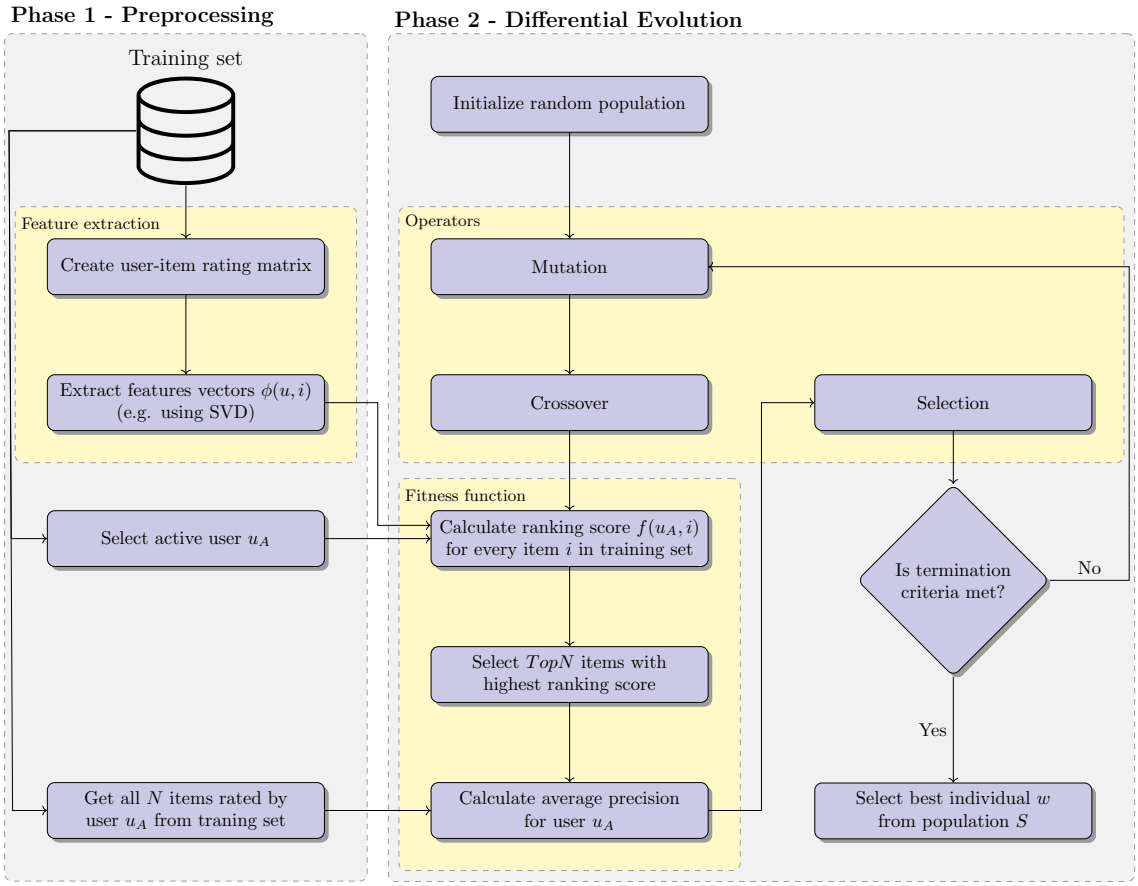

Fig. 2. System architecture.

4. The proposed RecRankDE method

In this section we will show you how to apply the learning to rank in the recommendation systems. Based on chapter 3.3, in our system, users will be marked as U and will correspond to queries Q, and items will be marked as I and will correspond to documents D. The aim of our RecRankDE method is, on the basis of the training set, to discover such a ranking function that generalizes information about the user well, helping to predict his next choices in the system. The RecRankDE method firstly adopts a set of training data T, which contains user-item pairs (u, i) with their corresponding feature vector $\phi(u, i)$. The method then uses the fitness function described in section 4.1 and generates new populations of individuals until it reaches the stop criterion. The scheme of the system is presented in Fig. 2.

4.1. Fitness function

The key element of the Differential Evolution algorithm is the fitness function. Because of this function, after using mutation and crossover operators, we can determine the quality of a new individual in relation to an individual from the previous generation. Owing to this solution, only the best adapted individuals will get to the next generation. In our case, the individual is a vector of real numbers, where the dimension of such a vector corresponds to the number

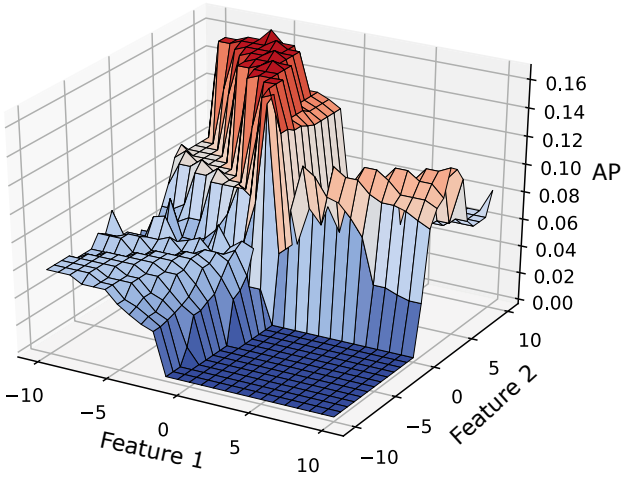

Fig. 3. Example visualisation of fitness landscape if every item in system is described by two features.

of features describing items. The fitness function is calculated for the selected user in the system and can be described in the following way:

$$Fitness = AP(w, T, N, u_A) \tag{8}$$

where w specifies an individual in the DE algorithm, in this case the weight vector. Formula 8 means the average precision that would be calculated on the training set T for the selected active user u_A, using the weight vector w in the ranking function $f(u_A, i)$ (formula 5), which assigns a value to each item i in training set T for a given user u_A. The parameter N is also passed to this function, which means the number of rated items by user u_A in the training set T. In our system $TopN$ in fitness function is equal to N. Visualisation how fitness landscape could look like if every item in system would have only two features is presented in Fig. 3

5. Experiments

A popular approach to the evaluation of the recommendation system is to anticipate what kind of grade the user will give to a given item (movie). However, such an approach does not fully correspond to how recommendation systems actually work, where recommendations are displayed to the user in the form of a list of proposed items. In addition, the order of displayed products is also important, as the top of the list there should be the products that the user is most likely to be interested in. This is not a trivial problem due to the size of the collection of items from which we have to choose the very videos that will be of interest to the user.

5.1. Experiments setup

Experiments conducted in this article were performed using the popular MovieLens database [14]. The database has 100,000 ratings based on a scale of 1-5 by 943 users for 1682 films. Each user has rated at least 20 movies, and incomplete data has been removed. In order for our RecRankDE algorithm to create a ranking, it is required that each item in the system is described by a set of features. Although some movie features (e.g. genres) are already available in the MovieLens data set, we will use the SVD algorithm described in chapter 3.2 to generate latent features on which our algorithm will learn a given user's preferences. The RecRankDE algorithm has been implemented in two programming languages. In Python the research preparation and results analysis have been implemented, and to speed up calculations the entire Differential Evolution algorithm has been implemented in C#. Such a solution allowed to

use the performance of C# to perform time-consuming calculations, while the Python, owing to a large number of available libraries, greatly facilitated the preparation of the environment needed to conduct reliable research. The research was conducted on a computer with Intel Core i5-7600 processor with a 3.50GHz frequency and 16GB RAM.

5.2. Research methodology

First, the ratings for each user were sorted by timestamp and then divided into two sets: train (80%) and test (20%). Naturally, this approach is completely justified, because based on the user's previous interactions with the system, we want to predict his future behavior. In order to check the quality of the generated recommendations the evaluation protocol of "all items" described in [20] was used. This means that in order to check the quality of generated recommendations our system had to select *TopN* items, from all the items that occurred in the training set and which the user has not yet rated. For example, if there are 1600 items in the training set and the user has already rated 100 of them, the system has to recommend *TopN* items (e.g. 10) from a set of 1500 items. The quality of the generated recommendations for a given user is determined by the average precision (AP). This measure compares *TopN* of recommended items suggested by our system with items that a given user has in his/her test set. Due to the time-consuming process of learning the ranking function through Differential Evolution, 30 users were randomly selected and the recommendations were generated for them and then these results were averaged. To compare the quality of the generated recommendations a regular SVD algorithm marked as "PureSVD" and an algorithm using the logistic loss function marked as "Logistic" were used. Additionally, in order to demonstrate how low quality can be achieved without using any technique, the quality of generated recommendations was calculated by randomly selecting items from the training set (marked as "Random") and the most frequently selected items by users that were present in the training set (marked as "Most Popular"). During the research it was noticed that the quality of the generated recommendations is strongly influenced by the number of items rated by a given user. Therefore, the research focused on this element of the system. In addition, the Wilcoxon test was conducted to demonstrate the statistical significance of the presented results. The parameters of the Differential Evolution algorithm were taken from our previous work [1] and are presented in Tab.1.

Table 1. Differential Evolution parameters.

Parameter name	Value
Population	25
Number of Iterations	250
Crossover's Probability	0,9
Amplification Factor F	0,8
Dimensions (number of latent features)	15

6. Results

Analyzing the results presented in Tab. 2 (*TopN=10*), it can be seen that when the minimum number of the rated items was 50 and 100, the quality of recommendations generated by the RecRankDE algorithm was lower compared to the pure SVD algorithm. However, since 150 items were rated, the quality of the generated recommendations expressed by MAP started to increase significantly, which is shown in Fig. 4. The values of *Random* and *Most Popular* are very low and were omitted in this chart. In addition, research was conducted to check the quality of the generated recommendations depending on the number of items that are recommended for the user (MAP@), and the results are presented in Tabs. 3, 4 and 5.The results that were statistically significant (p-value<0.05) were underlined.

7. Conclusions and future work

In this article we have presented how the Differential Evolution algorithm can be used to generate personalized recommendations for a given user. The proposed algorithm has been tested on the public database MovieLens. Exper-

Table 2. The quality of generated recommendations (MAP) depending on the minimum amount of movies that the user has rated in the data set.

Min rated Items	Random	Most Popular	PureSVD	Logistic	RecRankDE
50	0,002	0,022	**0,093**	0,044	0,081
100	0,001	0,020	**0,116**	0,063	0,107
150	0,008	0,021	0,152	0,132	**0,245**
200	0,018	0,018	0,134	0,093	**0,231**
250	0,022	0,022	0,158	0,183	**0,270**
300	0,012	0,025	0,134	0,157	**0,297**

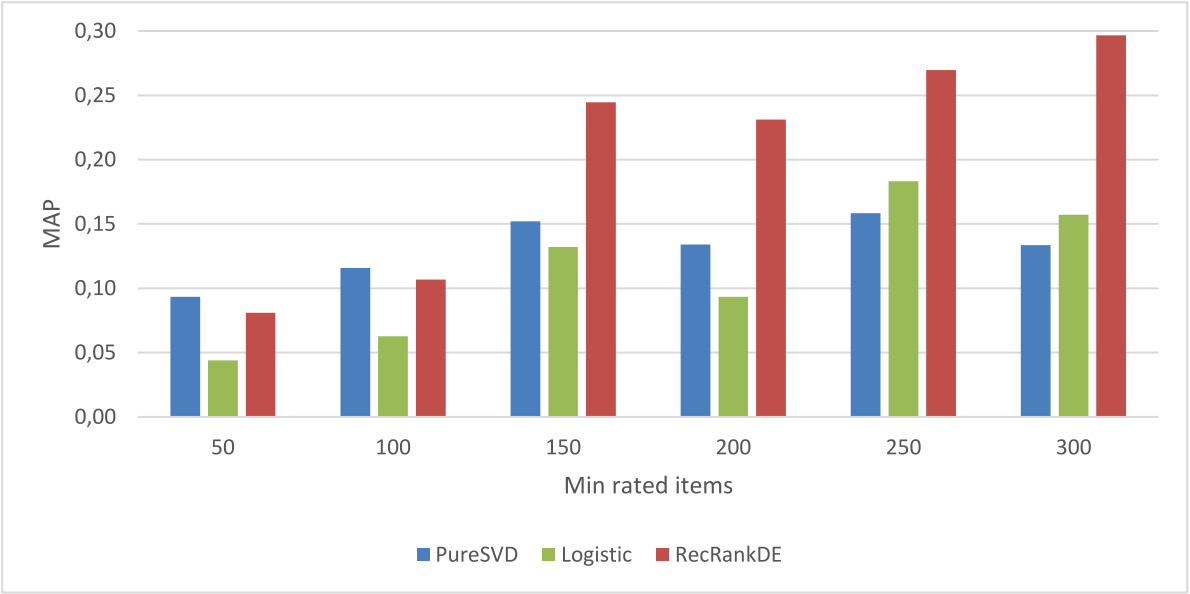

Fig. 4. The quality of generated recommendations (MAP) depending on the minimum amount of movies that the user has rated in the data set.

Table 3. The quality of the recommendations (MAP) generated for users who have rated at least 100 movies.

MAP@	Random	Most Popular	PureSVD	Logistic	RecRankDE
1	0,033	0,067	0,200	0,133	**0,300**
2	0,017	0,050	0,183	0,100	**0,258**
3	0,030	0,044	0,141	0,089	**0,245**
4	0,019	0,040	0,147	0,081	**0,190**
5	0,016	0,034	0,152	0,085	**0,199**
6	0,021	0,030	0,138	0,076	**0,172**
7	0,007	0,025	**0,133**	0,071	0,128
8	0,010	0,024	**0,129**	0,067	0,125
9	0,012	0,022	**0,120**	0,067	0,107
10	0,011	0,024	**0,116**	0,063	0,115

iments and their analysis show that our algorithm significantly improves the quality of generated recommendations, which was confirmed by statistical tests. It was also presented how the number of items rated by the user in the system significantly influences the results of the tests. In future works we will focus on multi-criteria optimization. In addition, we will work on performance improvement of our algorithm, so that the process of learning user preferences can take place in the shortest possible time and we will be able to examine larger datasets. We will also implement other metaheuristic algorithms to compare their performance with our approach.

Table 4. The quality of the recommendations (MAP) generated for users who have rated at least 200 movies.

MAP@	Random	Most Popular	PureSVD	Logistic	RecRankDE
1	0,100	0,000	0,367	0,233	**0,467**
2	0,000	0,025	0,225	0,200	**0,458**
3	0,007	0,035	0,213	0,170	**0,404**
4	0,033	0,026	0,197	0,151	**0,329**
5	0,015	0,024	0,175	0,129	**0,292**
6	0,013	0,024	0,160	0,118	**0,301**
7	0,021	0,023	0,156	0,108	**0,277**
8	0,023	0,020	0,145	0,100	**0,235**
9	0,007	0,020	0,136	0,099	**0,273**
10	0,015	0,021	0,134	0,093	**0,247**

Table 5. The quality of the recommendations (MAP) generated for users who have rated at least 300 movies.

MAP@	Random	Most Popular	PureSVD	Logistic	RecRankDE
1	0,033	0,100	0,300	0,300	**0,367**
2	0,042	0,050	0,192	0,275	**0,425**
3	0,032	0,048	0,161	0,239	**0,383**
4	0,031	0,036	0,158	0,231	**0,401**
5	0,038	0,032	0,151	0,210	**0,315**
6	0,027	0,030	0,150	0,195	**0,363**
7	0,018	0,030	0,150	0,176	**0,310**
8	0,026	0,026	0,140	0,164	**0,304**
9	0,034	0,024	0,133	0,163	**0,309**
10	0,014	0,024	0,134	0,157	**0,288**

References

[1] Bałchanowski, M., Boryczka, U., Dworak, K., 2018. Using differential evolution with a simple hybrid feature for personalized recommendation, in: Nguyen, N.T., Hoang, D.H., Hong, T.P., Pham, H., Trawiński, B. (Eds.), Intelligent Information and Database Systems, Springer International Publishing, Cham. pp. 137–146.

[2] Belém, F., Santos, R., Almeida, J., Gonçalves, M., 2013. Topic diversity in tag recommendation, in: Proceedings of the 7th ACM Conference on Recommender Systems, Association for Computing Machinery, New York, NY, USA. p. 141–148. doi:10.1145/2507157.2507184.

[3] Belém, F.M., Martins, E.F., Almeida, J.M., Gonçalves, M.A., 2014. Personalized and object-centered tag recommendation methods for web 2.0 applications. Inf. Process. Manag. 50, 524–553.

[4] Belém, F., Martins, E., Pontes, T., Almeida, J., Gonçalves, M., 2011. Associative tag recommendation exploiting multiple textual features, pp. 1033–1042. doi:10.1145/2009916.2010053.

[5] Bennett, J., Lanning, S., Netflix, N., 2007. The netflix prize, in: In KDD Cup and Workshop in conjunction with KDD.

[6] Bollegala, D., Noman, N., Iba, H., 2011. Rankde: Learning a ranking function for information retrieval using differential evolution, pp. 1771–1778. doi:10.1145/2001576.2001814.

[7] Boryczka, U., Juszczuk, P., Kłosowicz, L., 2009. A comparative study of various strategies in differential evolution, in: Evolutionary Computing and Global Optimization, pp. 19–26.

[8] Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G., 2005. Learning to rank using gradient descent, pp. 89–96. doi:10.1145/1102351.1102363.

[9] Cohen, W.W., Schapire, R.E., Singer, Y., 1999. Learning to order things. Journal of Artificial Intelligence Research 10, 243–270. doi:10.1613/jair.587.

[10] Diaz-Aviles, E., Mihai, G., Nejdl, W., 2012. Swarming to rank for recommender systems doi:10.1145/2365952.2366001.

[11] Diaz-Aviles, E., Nejdl, W., Schmidt-Thieme, L., 2009. Swarming to rank for information retrieval, in: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, Association for Computing Machinery, New York, NY, USA. p. 9–16. doi:10.1145/1569901.1569904.

[12] Freund, Y., Iyer, R., Schapire, R.E., Singer, Y., 2003. An efficient boosting algorithm for combining preferences. J. Mach. Learn. Res. 4, 933–969.

[13] Guimarães, A.P., Costa, T.F., Lacerda, A., Pappa, G.L., Ziviani, N., 2013. Guard: A genetic unified approach for recommendation. JIDM 4,

295–310.
[14] Harper, F.M., Konstan, J.A., 2015. The movielens datasets: History and context. ACM Trans. Interact. Intell. Syst. 5, 19:1–19:19.
[15] Horvath, T., de Carvalho, A., 2016. Evolutionary computing in recommender systems: a review of recent research. Natural Computing doi:10.1007/s11047-016-9540-y.
[16] Karatzoglou, A., Baltrunas, L., Shi, Y., 2013. Learning to rank for recommender systems, pp. 493–494. doi:10.1145/2507157.2508063.
[17] Koren, Y., Bell, R., Volinsky, C., 2009. Matrix factorization techniques for recommender systems. Computer 42, 30–37.
[18] Liu, T.Y., 2009. Learning to rank for information retrieval. Found. Trends Inf. Retr. 3, 225–331. doi:10.1561/1500000016.
[19] Schafer, J.B., Frankowski, D., Herlocker, J., Sen, S., 2007. Collaborative filtering recommender systems. in: The adaptive web. lecture notes in computer science, volume 4321. chapter Collaborative Filtering Recommender Systems, pp. 291–324.
[20] Sidana, S., Trofimov, M., Horodnitskii, O., Laclau, C., Maximov, Y., Amini, M.R., 2017. Representation learning and pairwise ranking for implicit feedback in recommendation systems. arXiv:1705.00105.
[21] Storn, R., Price, K., 1997. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. J. of Global Optimization 11, 341–359.