Combining Cmab with Matrix Factorization and Clustering For Enhanced Movie Recommendations

Shuyue Xiong

School of Computer Science, Beijing University of Technology, Beijing, 100124, China

Abstract. The contextual multi-armed bandit (CMAB) algorithm faces problems of a large-scale, sparse data matrix and many arms. This paper proposes a method that combines matrix factorization with a clustering algorithm to enhance the performance of the CMAB algorithm. For the MovieLens 1M dataset, data processing is conducted through matrix factorization to optimize the algorithm's input features, and a clustering method is used to decrease the number of arms, thereby improving the recommendation efficiency and performance of the CMAB algorithm. During the experiment of the proposed method, Funk Singular Value Decomposition (FunkSVD) is employed to fill the user-movie sparse matrix with predicted ratings, then the SSE of K-means clustering algorithm is utilized to determine the optimal number of clusters, in which 3952 movies is clustered into 96 groups, significantly reducing the number of arms and exploration cost of the algorithm. The hyperparameter of Linear Upper Confidence Bound (LinUCB) is also optimized by achieving the lowest cumulative regret value. By comparing the performance of LinUCB with Upper Confidence Bound (UCB) and Thompson Sampling (TS), LinUCB notably outperforms the traditional Multi-Armed Bandit (MAB) algorithms in cumulative regret values, regret bucket statistics (approximately 650 times/1000 runs of recommending high-quality movies), and convergence slope (0.0718).

1 Introduction

1.1 Background

Recommendation systems play a key role in modern information services. To recommend the most appropriate information to different groups of users, the recommendation algorithms should be efficient enough. Methods such as deep learning models are widely used in recommendation systems, but they all face the problems of cold start and high computational cost [1]. The traditional MAB algorithms solve the cold start problem to a certain extent by dynamically balancing exploration and exploitation, providing real-time adaptability for recommendation systems, and they have been proven to apply to a variety of scenarios, not only limited to recommendation systems, but also including communications, finance,

xiongshuyue@emails.bjut.edu.cn

medical fields, and so on [2, 3]. However, traditional MAB algorithms, such as Explore-Then-Commit (ETC), UCB, and TS algorithms, cannot process contextual information and have poor processing capabilities for complex information; thus, the algorithms need to spend a lot of time on exploration, resulting in low algorithm efficiency [4, 5]. CMAB, such as LinUCB, can be used to solve the problems related to contextual information. However, LinUCB still faces the problems of the computational efficiency of feature extraction, resulting in poor performance of the algorithm when processing high-dimensional vectors of user and movie features [6].

1.2 Research Issues and Gaps

It has been shown that matrix factorization and clustering algorithms are helpful in reducing the computational complexity and overhead of the recommendation system [4, 7]. For example, FunkSVD is used as a collaborative filtering algorithm based on matrix factorization, and K-means is widely used in clustering algorithms. The two methods have different effects and contributions to improving the algorithm efficiency and recommendation effect of LinUCB by processing the data collaboratively. But, how to reasonably combine them to achieve the best performance of the LinUCB algorithm is still an open problem that has not been fully explored.

1.3 Methods and Contributions

This paper explores the impact of matrix factorization and clustering algorithms on the efficiency improvement of the LinUCB algorithm and verifies it on the MovieLens 1M dataset. The following innovations fill the above research gaps.

Matrix factorization: FunkSVD is used to process the user-movie sparse matrix and predict the scores of movies that users have not rated.

Clustering algorithm: K-means is used to cluster all movies into suitable groups as arms of LinUCB according to movie features. The optimal number of clusters, K value, is determined by the sum of the squared errors (SSE) of the algorithm [2].

Comparative experiment: The data is first processed by FunkSVD and K-means methods, and then it is used in LinUCB, UCB, and TS algorithms, respectively. Evaluation indicators of these three algorithms' performance include average cumulative regret with the standard deviation, regret value bucket statistics, and convergence slope curve.

1.4 Paper Structure

The paper is organized as follows: the second section reviews relevant literature, the third section describes the proposed method in detail, the fourth section presents the experimental results, and the last section summarizes the entire paper.

2 Related Work

2.1 CMAB Algorithm in Recommendation System

In the movie recommendation system, the MovieLens 1M dataset contains a large amount of user information about age, gender, and occupation, as well as movie information about genres and ratings. This contextual information about the user and movies can be used to help LinUCB reduce exploration costs and improve overall performance. LinUCB can recommend suitable movies to users based on user and movie features, while traditional MAB algorithms, such as UCB and TS, can only recommend so-called high-scoring movies

to users and cannot make corresponding recommendations based on different user features. It can be seen from the pseudo code of LinUCB that both LinUCB and UCB are based on the exploration-exploitation framework of confidence intervals, but LinUCB introduces user and movie features when calculating confidence intervals. Furthermore, in contrast, the TS algorithm selects arms through probabilistic sampling [8].

Table 1 The LinUCB pseudo code

	Algorithm LinUCB algorithm
0:	Input: $\alpha \in \mathbb{R}_+$
1:	**for** t = 1, 2, 3, …, T **do**
2:	Observe features of all arms $a \in \mathcal{A}_t$: $\mathbf{x}_{t,a} \in \mathbb{R}^d$
3:	**for all** $a \in \mathcal{A}_t$ **do**
4:	**if** a is new **then**
5:	$\mathbf{A}_a \leftarrow$ Id (d-dimensional identity matrix)
6:	$\mathrm{b}_a \leftarrow$ Od×1 (d-dimensional zero vector)
7:	**end if**
8:	$\widehat{\boldsymbol{\theta}}_a \leftarrow \mathbf{A}_a^{-1}\mathbf{b}_a$
9:	$p_{t,a} \leftarrow \widehat{\boldsymbol{\theta}}_{t,a}^T \mathbf{x}_{t,a} + \alpha \sqrt{\mathbf{x}_{t,a}^T \mathbf{A}_a^{-1} \mathbf{x}_{t,a}}$
10:	**end for**
11:	Choose arm $a_t = \mathrm{argmax}_{a \in \mathcal{A}_t} p_{t,a}$ with ties broken arbitrarily, and observe a real-valued payoff r_t
12:	$\mathbf{A}_{a_t} \leftarrow \mathbf{A}_{a_t} + \mathbf{x}_{t,a_t}\mathbf{x}_{t,a_t}^T$
13:	$\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t \mathbf{x}_{t,a_t}$
14:	**end for**

The hyperparameter α, as shown in line 9 in Table 1, is an important value that affects the LinUCB algorithm's exploration-utilization trade-off and recommendation effects. It can be optimized by achieving the lowest cumulative regret value.

2.2 Matrix Factorization

The information about the user-item interaction history can be used to extract similarities and predict unknown ratings. Matrix factorization performs well in dealing with sparse matrices [9]. Through FunkSVD, the existing ratings in the sparse matrix of user-movie ratings are used to predict the scores of movies that the user has not rated, and then the entire user-movie matrix is fully filled with the predicted values [10]. This algorithm is suitable for large-scale sparse data and has stronger scalability than traditional Singular Value Decomposition (SVD).

2.3 Clustering Algorithm

K-means is one of the commonly used unsupervised algorithms in clustering [7]. It partitions data into K clusters by using the distance between the average points to calculate the proximity between the features and the centroid. It can efficiently cluster data according to the similarity of the features between the data, which can reduce the high computing cost of

large-scale data and improve the feasibility of the algorithm [7, 11]. In the field of recommendation systems, the K-means algorithm is used to cluster movies as arms in the LinUCB, which facilitates personalized recommendations and formulates precise marketing strategies.

2.4 Research Gap

In the movie recommendation scenario, there are few research works on reasonably and effectively combining matrix factorization and clustering algorithms with LinUCB. The hybrid algorithm proposed in this paper solves this research gap. In addition to exploring the performance of different algorithms in terms of cumulative regret value, this paper further explores the algorithm performance of different algorithms by other evaluation indicators, including the regret bucket and slope convergence curve.

3 Methodology

Figure 1 introduces the experimental procedure of this paper. First, the data of the MovieLens1M dataset is preliminarily processed, and FunkSVD is used to predict the rating matrix. Then, a comparative experiment is conducted to find the optimal experimental conditions, including K value as a number of clusters and α value of the LinUCB algorithm, and then experiments are conducted on the LinUCB, UCB1, and TS algorithms. Evaluation indicators include the average cumulative regret value with the standard deviation, regret bucket statistics, and the convergence of the slope of the cumulative regret curve.

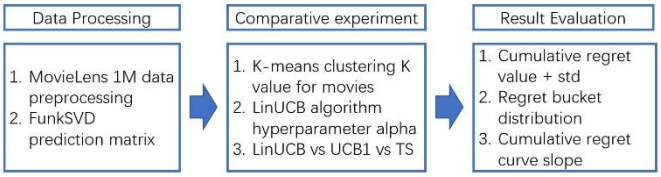

Figure 1 Experimental flow chart (Picture credit: Original)

3.1 Data Processing

The MovieLens 1M dataset contains a total of 1,000,209 rating records, with 6,040 users rating 3,952 movies, in which each user rates at least 20 movies. Each user, except for the user ID, has three features: *gender*, *age*, and *occupation*. *Age* is divided into 7 age groups and then normalized. *Occupation* includes 20 different kinds of occupations, which are expanded into 20 different features through one-hot encoding. Therefore, each user has a total of 22 features in addition to their user ID. Each movie, except for the movie ID, has two features: *genre* and *year*, and each movie may have multiple genres. *Genre* is also one-hot encoded and expanded into 18 different features, so each movie has a total of 19 features in addition to its movie ID.

3.2 FunkSVD Prediction Matrix

Before using FunkSVD to process sparse matrices, a dimensionality reduction method, such as principal component analysis (PCA), is originally considered to solve the problem, but dimensionality reduction by PCA is usually used to reduce tens of thousands of dimensional feature vectors to 100~200 dimensional feature vectors. On the other hand, the total number

of user features and movie features is only 41, which is a small number of features. As a result, PCA dimensionality reduction is not necessary. Finally, the FunkSVD algorithm is used here to directly optimize the user ratings based on the potential feature matrix of users and movies by gradient descent, and then fill in the blank ratings in the original user-movie matrix to generate a complete prediction matrix.

3.3 K-means Clustering

Before the formal experiment of the LinUCB algorithm, the preliminarily processed user and movie features were directly used in the algorithm, but it was found that no matter how many rounds were run, the accumulated regret value curve generated by the LinUCB algorithm could not converge and was always a linear curve. After preliminary analysis, the LinUCB algorithm treated all 3952 movies as independent arms, resulting in a small number of explorations for each arm, and was unable to correctly find the optimal arm, which caused the algorithm to continue exploring and to be inefficient. This paper uses the K-means clustering method to first cluster and group the 3952 movies, and then finds the optimal clustering *K* value by the SSE, and finally clusters into 96 movie groups, greatly reducing the number of arms and improving the efficiency of the algorithm.

3.4 Experimental Design

a) FunkSVD is used to obtain the prediction matrix and the predicted score distribution.

$$min_{p,q} \sum_{(u,i)\in R} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \quad (1)$$

where p_u is a latent factor vector for user u (dimension = d), q_i is the latent factor vector for movie i (dimension = d), r_{ui} is the observed rating of user u for item i, and λ is a regularization parameter to prevent overfitting.

b) K value: The best K value in the K-means algorithm is determined by minimizing the SSE.

$$SSE = \sum_{i=1}^{k} \sum_{x\in C_i} \|x - \mu_i\|^2 \quad (2)$$

where *k* is the number of clusters, C_i represents the *i*-th cluster, μ_i is the centroid of cluster C_i, and x denotes a data point in the cluster. Different numbers of K are then compared to analyze their impact on LinUCB's cumulative regret.

c) α value in the LinUCB: α is a hyperparameter that affects LinUCB exploration. The impact of different α values on the cumulative regret of the LinUCB algorithm is compared, and finally α is set to 0.05.

d) Comparative experiment 1: Compare the different performances of average cumulative regret with the standard deviation (std) obtained by LinUCB, UCB1, and TS algorithms while using the same data. Equation (3) shows how to calculate the cumulative regret.

$$R(T) = \sum_{t=1}^{T} (\mu^* - \mu_{a_t}) \quad (3)$$

where μ^* is the reward of the optimal action, μ_{a_t} is the reward of the chosen action at time t, and T is the total time of the whole experiment (e.g., T=5000 runs).

e) Comparative experiment 2: Compare the statistical changes of regret buckets obtained by the three algorithms.

f) Comparative experiment 3: Compare the regret convergence slope curves obtained by the three algorithms based on equation (4).

$$Slope(t) = \frac{R(t) - R(t - \Delta t)}{\Delta t} \tag{4}$$

where Δt means the windows of time (e.g., Δt = 100 steps).

4 Results and Analysis

4.1 FunkSVD Prediction Matrix

The FunkSVD algorithm is used to predict and fill the user-movie sparse matrix according to equation (1). Then, user-movie matrix score statistics are obtained as shown in Table 2, which summarizes the distribution of ratings in the prediction matrix.

Table 2 User-movie matrix score statistics

Categories	Rating	Proportion
Excellent	5	3.8%
	4	30.1%
Good	3	42.1%
Poor	2	19.3%
Terrible	1	4.7%

The user's evaluation of the movie is divided into 4 levels, and it is found that the proportion of users who rated the movie as Excellent and Good is very high, almost 76%. Thus, it is inevitable to recommend some Good movies to users because of score distribution, causing regret accumulation.

4.2 Impact of K Value on The Algorithm

Clustering movies means recommending a category of movies to a user. The K-means clustering algorithm clusters movies based on their different features, and then uses the features of the centroid of the cluster as the features of the cluster movies for algorithm calculations. The number of clusters will affect the accumulation of cumulative regret and the performance of the algorithm. The SSE between all data points and the centroid of the cluster to which they belong is calculated using the SSE (equation (2)). The inflection point where the acceleration drops sharply, as shown in Figure 2, is the optimal K value.

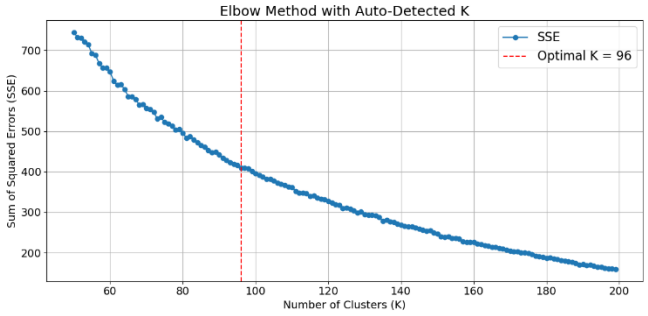

Figure 2 Optimal K value (Picture credit: Original)

As shown in Figure 2, the optimal K value is 96. To verify the conclusion, the LinUCB algorithm was tested 10 times by changing the K value, and each experiment was run for 5000 rounds. The average cumulative regret and errors were calculated according to equation (3), and the results are recorded in Table 3 below.

When K is 96, the algorithm indeed obtained a smaller average cumulative regret, and the algorithm performed relatively better. By contrast, K=60 means that there are too few clusters, and the movies features within the cluster may have low similarity. Users have different ratings for different movies in the class, so that the algorithm cannot find the optimal cluster for the user; K=120 means that the number of clusters is too large, which may cause serious overlap between clusters and lead to a large number of arms, so that the algorithm needs to explore more arms. As can be seen from the results, both situations will cause the accumulation of cumulative regret and affect the efficiency of the algorithm.

Table 3 Impact of different K values on LinUCB

K value	Average cumulative regret	std
60	668.15	8.92
96	579.96	25.42
120	607.73	35.88

4.3 Impact of α value on the algorithm

As a hyperparameter for calculating confidence intervals in LinUCB, the parameter α directly affects the algorithm's exploration-utilization trade-off and recommendation effect. The larger the α value, the more the algorithm tends to explore different arms. If α value is too large, the algorithm will frequently explore uncertain arms, resulting in long-term and rapid accumulation of cumulative regret. The smaller the α value, the lower the degree of algorithm exploration. But if it is too small, it may fall into a local optimum, affecting the algorithm's inability to explore the optimal arm. Due to the processing of data by FunkSVD and K-means algorithms, the LinUCB algorithm's demand for exploration has been reduced to a certain extent, and α value can be set to a smaller value.

Table 4 Impact of different α values in LinUCB

α value	Average cumulative regret	std
0.1	601.21	43.78
0.05	575.38	55.42
0.01	610.27	59.04

Changing the α value, such as in the experiment to explore the K value, repeat the LinUCB algorithm 10 times, each time running 5000 rounds. In Table 4, by comparing different α values, the performance of the LinUCB algorithm in terms of average cumulative regret is best when α is set to 0.05.

4.4 Performance Comparison of LinUCB, UCB1, and TS Algorithm

4.4.1 Average Cumulative Regret

LinUCB, UCB1, and TS algorithms used the same data processed by FunkSVD and K-means, respectively, to conduct two experiments. Each experiment was conducted 10 times, with 5,000 rounds each time. According to equation (3), if the regret is a positive number, it is accumulated. If it is a negative number, no regret is generated in that round. The best reward

is obtained by the average reward of nearly 100 rounds. Finally, the average cumulative regret and standard deviation of the 10 experiments were obtained.

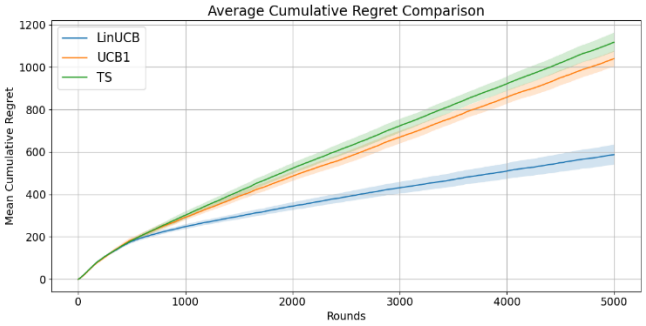

Figure 3 Comparison of cumulative regret values of the three algorithms in 5,000 rounds (Picture credit: Original)

It can be seen from Figure 3 that the LinUCB algorithm performs better than the UCB1 and TS algorithms, with a smaller average cumulative regret value, a flatter curve, and a slower regret accumulation rate, indicating that LinUCB finds the optimal arm faster and can recommend the best movie category for the user based on their different characteristics. The average cumulative regret values of the UCB1 and TS algorithms are larger, the slope of the curve is large, and the accumulation rate is fast, so the performance of both algorithms is a little poor. Furthermore, the number of runs per round is also increased to 10,000 times, and the results of both 5,000 runs and 10,000 runs are recorded in Table 5 and Table 6. It can be seen that the LinUCB algorithm still performs well. These results show that LinUCB uses user-movie features as contextual information to make the algorithm perform better than traditional non-contextual MAB.

Table 5 Accumulated regret values of 5,000 runs

	Average cumulative regret	std
LinUCB	567.76	28.46
UCB1	1035.55	26.49
TS	1084.18	36.51

Table 6 Accumulated regret values of 10,000 runs

	Average cumulative regret	std
LinUCB	899.22	57.10
UCB1	1909.57	45.38
TS	2135.44	110.22

4.4.2 Regret Bucket Distribution

Since there are only 5 kinds of values for the rating, the value range of best-reward after cold start is 0.65~0.72. It means that even if the algorithm finds the best arm, if the user-movie prediction rating is not 5 or 4 in the round, it will cause a large accumulation of regret value. According to Table 1, 42.1% of the ratings are 3 points, which means that even if it is the best arm, a movie with a rating of 3 points will still be drawn as an arm, resulting in the regret curves of the three algorithms always being linear, rather than the ideal log-type. Therefore,

the recommendation effect of different algorithms cannot be intuitively seen only from the average cumulative regret graph. For this reason, as shown in Table 7, four buckets are formed according to the different regret values, and the number of different regret values generated by selecting different arms in each 1000 rounds of the algorithm is counted respectively, and then it is judged whether the algorithm recommends the best movie category to the user as many as possible.

Table 7 Bucketing basis

Categories	Rating	Regret
Excellent	5 (1.0) 4 (0.75)	-1 ~ 0.1
Good	3 (0.5)	0.1 ~ 0.3
Poor	2 (0.25)	0.3 ~ 0.5
Terrible	1 (0.0)	0.5 ~ 1

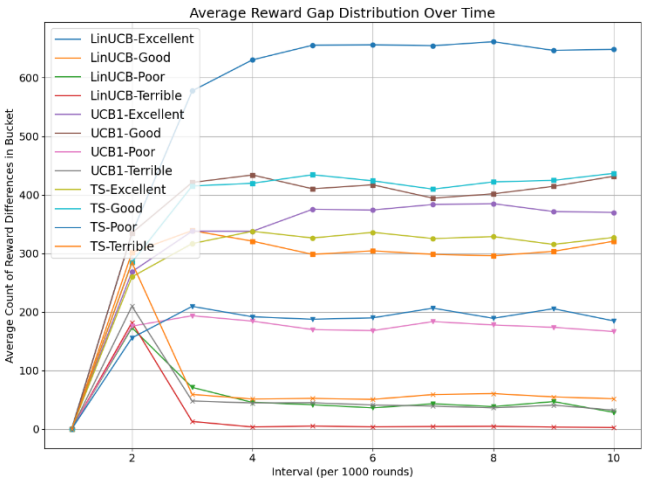

Figure 4 Comparison of different algorithms' regret bucket distribution (Picture credit: Original)

As can be seen from Figure 4, from round 5000 to round 10000, LinUCB recommended excellent and high-quality movie clusters about 650 times in every 1000 rounds, which is significantly more than UCB1 and TS algorithms, and rarely recommended poor and terrible movie clusters. In addition, the LinUCB algorithm significantly recommended movies with an estimated score of 4 and 5 points to users after 3000 rounds. Observing the bucket curves of UCB1 and TS, it can be seen that the algorithms often recommend good movies with a predicted score of 3 points to users compared to high-quality movies with a score of 4 and 5 points, and recommend poor and terrible movies more than LinUCB, resulting in more accumulated regret. Among the three algorithms, the LinUCB algorithm performed the best, and the TS algorithm performed the worst. It can be confirmed that the LinUCB algorithm can recommend the best movie for different users more quickly by utilizing user-movie feature information. In addition, it is unexpected that the TS algorithm is the worst-performing of the three algorithms. This shows that the TS algorithm has certain limitations when facing discrete scores of non-continuous divisions and concentrated scores (about 72% of the scores are 3 or 4 points). The random sampling of TS may repeatedly select movie clusters with the same score, causing the TS algorithm to accumulate a large amount of regret and perform the worst.

4.4.3 Convergence Curve Slope

In order to explore the convergence speed of the three algorithms, the slopes of the average cumulative regret convergence curves of the three algorithms in 10000 rounds are calculated respectively according to equation (4), and the following figure is obtained.

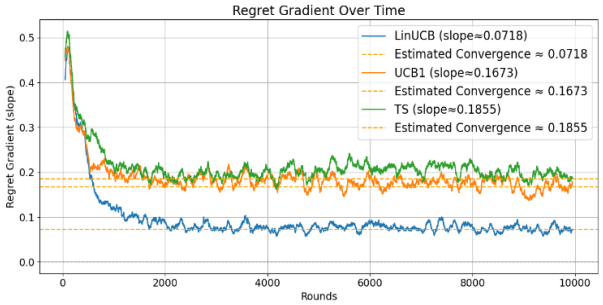

Figure 5 Slope of different algorithms' regret curve (Picture credit: Original)

The closer the cumulative regret curve value is to zero, the closer the algorithm is to convergence, and the exploration-utilization balance is reached. The difference between the potential benefits brought by exploration and the actual benefits of utilization is gradually reduced, and the algorithm performs better. From Figure 5, it can be found that the estimated slope convergence value of LinUCB is 0.0718, which is significantly smaller than the values of UCB1 and TS algorithms. Thus, the curve fluctuation is smaller, indicating that the LinUCB algorithm can more stably select the optimal arm, which means the algorithm is closer to convergence, and the performance is better.

5 Conclusion

The experiment first explored the data processing aspects, such as the score distribution of the FunkSVD prediction matrix and the impact of the number of K-means clusters on the LinUCB algorithm, and found the optimal number of clusters. Next, the optimal value of the hyperparameter of the LinUCB algorithm was determined. After that, the performance of LinUCB, UCB1, and TS algorithms was compared. It was found that compared with TS algorithms, LinUCB avoided the huge impact of discrete scores (scores have only 5 fixed values) and concentrated scores (scores are concentrated in 3 and 4 points) on algorithm sampling by using confidence intervals like TS algorithm random sampling; compared with traditional UCB1, LinUCB can quickly and stably reach convergence by combining user-movie context information features, and recommend the best movie clusters for different users. In summary, the FunkSVD algorithm and K-means clustering assist the LinUCB algorithm in processing large-scale samples and improve the efficiency of the algorithm. In the context of CMAB, the LinUCB algorithm can obtain significantly better performance than traditional non-context MAB, such as UCB1 and the TS algorithm. Next, a good neural network can be used as a type of supervised learning and combined with CMAB to help the algorithm further improve algorithm efficiency and performance.

References

1. Zhao J.: 'Comparison of multi-armed bandit algorithms in advertising recommendation systems', Applied and Comput-ational Engineering, 2024, 83, pp. 62-71.

2. Pilani A., Mathur K., Agrawald H., et al.: 'Contextual bandit approach-based recommendation system for personalized web-based services', Applied Artificial Intelligence, 2021, 35, (7), pp. 489-504
3. Bouneffouf D., Rish I., Aggarwal C.: 'Survey on applications of multi-armed and contextual bandits'. Proc. IEEE Congress on Evolutionary Computation (CEC), 2020, pp. 1-8.
4. Chen Y.: 'Contextual bandits to increase user prediction accuracy in movie recommendation system'. Proc. ITM Web Conf., 2025, 73, pp. 1018.
5. Silva N., Werneck H., Silva T., et al.: 'Multi-armed bandits in recommendation systems: A survey of the state-of-the-art and future directions', Expert Systems with Applications, 2022, 197, pp. 116669
6. Yan C., Han H., Zhang Y., et al.: 'Dynamic clustering-based contextual combinatorial multi-armed bandit for online recommendation', Knowledge-Based Systems, 2022, 257, pp. 109927
7. Bandyopadhyay S., Thakur S., Mandal J.K.: 'Product recommendation for e-commerce business by applying principal component analysis (PCA) and K-means clustering: benefit for the society', Innovations in Systems and Software Engineering, 2021, 17, (1), pp. 45-52
8. Elena G., Milos K., Eugene I.: 'Survey of multiarmed bandit algorithms applied to recommendation systems', International Journal of Open Information Technologies, 2021, 9, (4), pp. 12-27
9. Xu S.: 'BanditMF: Multi-armed bandit based matrix factorization recommender system', arXiv:2106.10898, 2021
10. Klymash M., et al.: 'Model of large sparse datasets processing efficiency in IIOT'. Proc. 17th Int. Conf. Experience of Designing and Application of CAD Systems (CADSM), 2023, 1, pp. 45-49.
11. Jayalakshmi S., Ganesh N., Čep R., et al.: 'Movie recommender systems: Concepts, methods, challenges, and future directions', Sensors, 2022, 22, (13), pp. 4904.