Dynamic Programming Based User Clustering For Recommendation System

***Abstract*—**This work proposes the application of dynamic programming to create cluster(s) of like-minded users in groups to trigger mutual recommendations among them for targeting specific products in order to maximize revenue generation.

***Keywords*—**User Clustering, Dynamic Programming, Collaborative Filtering, K-Means Clustering

I. INTRODUCTION

If we are able to gather users in a group, it becomes beneficial and easier for us to recommend them products that their peers are using. Through this, we can also address the outlier users in minority group [1]. One research area for recommendation systems is the algorithms used for clustering the users. Collaborative Filtering is used for establishing similarity among users based on the products they like. If two users are similar as per their choices and one of them likes some extra product, that product may be recommended to another user [2].

One application of dynamic programming is Sum Problem where we have to find different combinations of numbers that add up to give a number. The different combinations can become the group of users. This has been elaborated in the following sections.

II. RELATED WORKS

Phorasim et. al [3] use K-Means Clustering to categorize based on their interests and establish similarity using Pearson Correlation Coefficient and Euclidian distance. Garanayak et. al [4] use item based collaborative filtering after using K-Means Clustering.

III. METHODOLOGY

Suppose we have to find fifth term of the fibonnaci sequence. We notice that –

F(0) = 0

F(1) = 1

F(2) = F(1) + F(0)

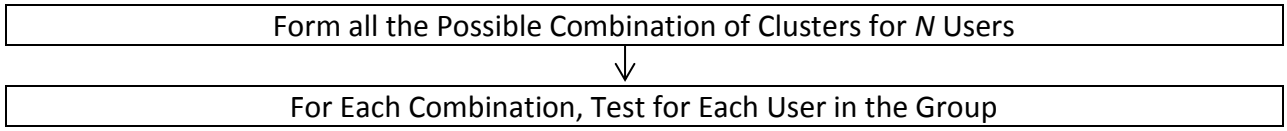
F(3) = F(2) + F(1)

F(4) = F(3) + F(2)

Each term is the sum of last two terms. Also, some terms on the right hand side above are repeating. Instead of calculating these overlapping sub problems, we can memorize their result and substitute.

In a recommendation system, we have to group users. Suppose there are 4 users, different ways they can be grouped as are: (1, 1, 1, 1), (1, 3), (2, 2), and (4).

Through Collaborative Filtering, similarity between the users can be found for each permutation and the clusters can be formed.



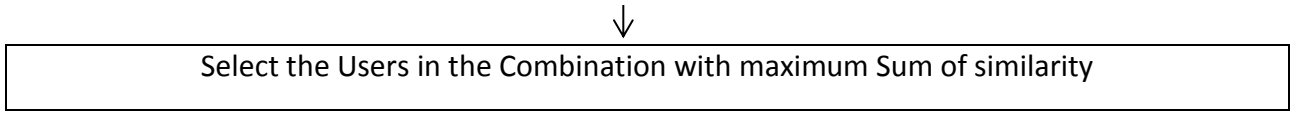


Figure 1. Process Flow

IV. ILLUSTRATION AND IMPLEMENTATION

Illustration

Suppose there are 4 Users giving a combination (1, 1, 1, 1), (1, 3), (2, 2), and (4).
And, the similarity matrix among them is as follows.

	User 1	User 2	User 3	User 4
User 1	1	0	3	2
User 2	2	3	1	4
User 3	0	4	2	1
User 4	3	4	1	1

Figure 2. Similarity Matrix

Now, we try all the combinations as follows.
Case 1: Combination (1, 1, 1, 1), where there are 4 Clusters having 1 User each. We do not find any worthwhile similarity to put all users in a group.
Case 2: Combination (1, 3) where there 2 Clusters having 1 User in 1 Cluster and 3 others in another Cluster. Now the question arises, "Which 3 Users are to be put together and which 1 to be left out?"
Therefore, we try combinations for this category using the similarity matrix given above.
(User 1, User 2, User 3) -> Similarity between User 1 and User 2 + Similarity between User 2 and User 3 + Similarity between User 1 and User 3 = 0 + 1 + 3 = 4
(User 2, User 3, User 4) -> Similarity between User 2 and User 3 + Similarity between User 3 and User 4 + Similarity between User 2 and User 4 = 1 + 1 + 4 = 6
(User 1, User 2, User 4) -> Similarity between User 1 and User 2 + Similarity between User 2 and User 4 + Similarity between User 1 and User 4 = 0 + 4 + 2 = 6
(User 1, User 3, User 4) -> Similarity between User 1 and User 3 + Similarity between User 3 and User 4 + Similarity between User 1 and User 4 = 3 + 1 + 2 = 6
Therefore, for Case 2, any 3 Users out of (User 1, User 2, User 4), and (User 1, User 3, User 4) can be put into 1 Cluster and remaining 1 into another.
Similarly, we can try for other Combinations (2, 2), and (4) i.e., **Case 3** and **Case 4**.

Implementation

```
import copy
clusters = {
        0: [[0]],
        1: [[1]]
```

```
}

def add_num_to_dict(num):
        global clusters
        running_list = []
        for i in range(1, num+1):
                for combos in clusters[num-i]:
                        for p in combos:
                                running_list.append([i,p])
        clusters[num] = [running_list]

num_of_users = int(input("Enter the number of User(s): "))
print("\n")

for num in range(2, num_of_users+1):
        add_num_to_dict(num)

for key in clusters.keys():
        print("---- Clusters for", key, "User(s) are---------")
        for i in clusters[key]:
                print(i)
                print("Total Clusters: ", len(i), "\n")
```

Output

```
Enter the number of User(s): 5
---- Clusters for 0 User(s) are---------
[0]
Total Clusters:  1

---- Clusters for 1 User(s) are---------
[1]
Total Clusters:  1

---- Clusters for 2 User(s) are---------
[[1, 1], [2, 0]]
Total Clusters:  2

---- Clusters for 3 User(s) are---------
[[1, [1, 1]], [1, [2, 0]], [2, 1], [3, 0]]
Total Clusters:  4

---- Clusters for 4 User(s) are---------
[[1, [1, [1, 1]]], [1, [1, [2, 0]]], [1, [2, 1]], [1, [3, 0]], [2, [1, 1]], [2, [2, 0]], [3, 1], [4, 0]]
Total Clusters:  8
```

---- Clusters for 5 User(s) are---------
[[1, [1, [1, [1, 1]]]], [1, [1, [1, [2, 0]]]], [1, [1, [2, 1]]], [1, [1, [3, 0]]], [1, [2, [1, 1]]], [1, [2, [2, 0]]], [1, [3, 1]], [1, [4, 0]], [2, [1, [1, 1]]], [2, [1, [2, 0]]], [2, [2, 1]], [2, [3, 0]], [3, [1, 1]], [3, [2, 0]], [4, 1], [5, 0]]
Total Clusters: 16

Complexity

The running time complexity of this implementation is $O(n^2)$.

V. CONCLUSION

Thus, this work illustrates how different clusters can be formed for onward implementation of K-Means Clustering which can serve as a future work.

VI. REFERENCES

[1] Misztal-Radecka J., Indurkhya B., "Bias-Aware Hierarchical Clustering for detecting the discriminated groups of users in recommendation systems", Information Processing & Management, Volume 58, Issue 3, 2021.

[2] Schafer, J.B., Frankowski, D., Herlocker, J., Sen, S., "Collaborative Filtering Recommender Systems", Springer, Berlin, pp. 291–324, 2007.

[3] Phorasim, Phongsavanh & Yu, Lasheng. (2017). Movies recommendation system using collaborative filtering and k-means. International Journal of Advanced Computer Research, volume 7. pp. 52-59, 2017.

[4] Garanayak, Mamata, et al. "Recommender System Using Item Based Collaborative Filtering (CF) and K-Means." International Journal of Knowledge-Based and Intelligent Engineering Systems, volume 2, pp. 93–101, 2019.