Chapter 4

OPHAencoder: An unsupervised approach to identify groups in group recommendations

This chapter is based on our journal article published in [68]. In this chapter, we employ one permutation hashing technique and an autoencoder model to compose groups automatically in group recommendations. One permutation hashing helps us to find a set of similar users. Similarly, employing an autoencoder model gives us the set of candidate pairs of users. The members of an automatically identified group are the union of users from both sets.

Outline: The rest of the chapter is organized as follows. The proposed model is described in Section 5.1. A comparative study between the proposed techniques and existing models is given in Section 6.2. Finally, Section 4.3 summarizes this work.

4.1 Proposed work

In the case of a recommendation task, a user gives ratings to items, and an unknown rating implies that there is no explicit information about the user's preferences for that item in a utility matrix. Since preference data acquisition is explicitly available in some rating scales, it is required to convert these ratings into '0' or '1' as one permutation takes binary values as inputs. In the case of the ml-100k or ml-1m dataset, the rating scale range is from 1 to 5. In the utility matrix, a user has provided ratings to items,

and zero signifies that a particular user does not give a rating to that item. Further, We apply MinMax scaler [1] to transform features by scaling each feature to a range of '0' and '1', and '0.4' is taken as the threshold. We take this threshold empirically. So, a value greater than or equal to the threshold is set to '1', and if it is less than the threshold, it is set to '0'. If we consider ml-100k or ml-1m dataset or Amazon-music, we apply the same method to normalize ratings from the user-item rating interaction matrix (utility matrix). Thus, representing a user-item rating matrix in the form of $(0, 1)$-matrix is considered an input of a characteristic matrix. We use KNN item-based collaborative filtering to justify this representation and predict unknown ratings. We calculate the predicted ratings of items using KNN item-based collaborative filtering and evaluate the score using RMSE (Root Mean Square Error).

In the case of the ml-100k dataset, using the original utility matrix, KNN item-based collaborative filtering gave the Mean RMSE score as 0.99. RMSE takes two vectors as input- original movie ratings and predicted movie ratings. In the case of our model, we obtained $(0, 1)$ - matrix, i.e., all the values in the matrix are either '0' or '1'. Here Jaccard similarity or hamming distance can be used to calculate the similarities between each pair of items. After calculating the similarity between each pair of items, KNN item-based collaborative filtering is applied to determine a user's predicted ratings for the items. Each item is a column vector of dimension 943×1. We weight the user's rating for each of these items by the similarities with the items' neighbours. Finally, we scale the prediction by the sum of similarities to get the final predicted rating. Prediction for an active user u on an item i is given as follows:

$$P_{u,i} = \frac{\sum_{\forall_j \in z_u(i)} (JS_{i,j} \times R_{u,j})}{\sum_{\forall_j \in z_u(i)} |(JS_{i,j})|} \tag{4.1}$$

Where the summation is over a set $z_u(i)$, i.e., the nearest neighbors of items which are similar to the target item i that the user u has rated. $JS_{i,j}$ is the Jaccard similarity between item i and j; $R_{u,j}$ is the original rating given by the user on the nearest neighbor of items. We take the weighted average of all the ratings on item i for user u for rating prediction.

Similarly, the quality of prediction is evaluated using the RMSE function. We get similar performance, and there is no significant deviation in the outcome. Item-based col-

[1] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

laborative filtering provides an RMSE score of 0.37 when considering normalized ratings in the $(0, 1)$-matrix. We compromise on accuracy as there is no well-established encoding standard for this conversion. After this, the One permutation hashing technique is applied to generate the signature matrix. In the case of one permutation hashing, the algorithm considers binary inputs only. We transpose this characteristic matrix such that columns represent the set of users. We further reduce this characteristic matrix using one permutation hashing, and it becomes easy to calculate the similarity between users in the obtained matrix.

4.1.1 One permutation hashing

One permutation hashing is a standard method for efficiently estimating set similarity in the presence of big data [124]. This property inspires us to find a set of similar users from the given dataset. The idea is straightforward; we shuffled the data (permutation using a hash function). It requires to compute only one hash function. Afterwards, data is divided into 'k' bins. We find the minimum index corresponds to '1' and store it as a hash value within each of these bins if there are no '1' in a particular bin, so the corresponding value is assigned to a letter E (Letter 'E' signifies empty). One permutation hashing uses only one permutation; permutation of a row is possible using a hash function in the form of: $h(x) = (ax + b) \text{mod} m$
where:

1. x is the row number of original characteristic matrix, i.e., $x \in 0, 1, 2, ..., n$.
2. a and b are any random numbers smaller or equal to the largest row number.
3. m is a prime number larger than the largest row number (n).

It breaks the space into 'k' equal bins while calculating one permutation hashing value. It requires re-indexing those bins to calculate the minhash values, i.e., the smallest non-zero position in each bin of the matrix. So, the signature matrix[2] consists of "total number of $'n'$ users $\times$ total numbers of bins" dimensions. Table 4.1 gives the minhash values after applying one permutation hashing over sets s_1, s_2, s_3, s_4. After employing one permutation matrix, our original characteristic matrix of size, 6×4 has reduced into a

[2]A reduced representation of a characteristic matrix after applying minhash technique

Table 4.1: one permutation hashing on four different sets

S_1	S_2	S_3	S_4	(2x+1)%7	$\pi(S_1)$	$\pi(S_2)$	$\pi(S_3)$	$\pi(S_4)$	OPH ($\pi(S_1)$)	OPH ($\pi(S_2)$)	OPH ($\pi(S_3)$)	OPH ($\pi(S_4)$)
0	1	1	0	1	1	0	0	0	0	E	1	1
1	0	0	0	3	1	0	1	1				
0	1	0	1	5	0	1	0	0	E	0	1	E
1	0	1	1	0	0	1	1	0				
1	0	0	1	2	0	1	0	1	1	0	E	0
0	1	0	0	4	1	0	0	1				

signature matrix of size 4×3. Equation 4.2 estimates the similarity between any two users, where $N_{mat}(X, Y)$ is the number of matching bins between the signature of two users, b represents the total number of bins composing the signature, and $N_{emp}(X, Y)$ refers to the number of matching empty bins.

$$SIM(X, Y) = \frac{N_{mat}(X, Y)}{b - N_{emp}(X, Y)} \tag{4.2}$$

Table 4.2 depicts user similarity scores from the obtained signature matrix. However, it is not directly applicable to the nearest neighbor search by building hash tables due to empty bins. Because of these empty bins, it is impossible to determine which bin value to use for bucketing. Many subsets are empty, and corresponding signature values are undefined.

Table 4.2: Estimate similarity between users in one permutation hashing

	b_0	b_1	b_2	SIM(S_3,S_4)
S_1	0	E	1	$\frac{1}{3-0}$=0.33 ($N_{emp} = 1$ if both S_i and S_j=Empty(E))
S_2	E	0	0	
S_3	1	1	E	
S_4	1	E	0	

Various densification algorithms have been proposed to resolve this problem. However, a new randomized technique called "Densification" (densifies the bins via random rotation [109, 110]) is used to combat this kind of problem efficiently. This issue is solved by performing rotation over one permutation hashing. Specifically, if one bin is empty, the hashed value from the first non-empty bin on the right (circular) is borrowed as the key of this bin, which supplies an unbiased estimate. Every empty bin is assigned the

Table 4.3: Densification in one permutation hashing

One permutation hashing(OPH)	Minhash values	Minhash values after densification (with D=3 & K=2)
OPH($\pi(S_1)$)	[0,E,1]	[0,1+C,1]=[0,4,1]
OPH($\pi(S_2)$)	[E,0,0]	[C,0,0]=[3,0,0]
OPH($\pi(S_3)$)	[1,1,E]	[1,1,1+C]=[1,1,4]
OPH($\pi(S_4)$)	[1,E,1]	[1,1+C,1]=[1,4,1]

value of the closest non-empty bin, towards the right (circular) with an offset C[3]. In [109], the authors provide an option to move either left or right with probability $\frac{1}{2}$. The offset is taken as $C = \frac{D}{k} + 1$, where D is the total number of bands and k is the total number of rows in a particular band during re-indexing of bins in one permutation hashing. Table 4.3 provides the minhash values of each set after densification.

To find similarity between two sets in a characteristic matrix; there are four possible combinations for comparison, i.e. : $|Type(1,1)| = x$, $|Type(1,0)| = y$, $|Type(0,1)| = z$ and, $|Type(0,0)| = d$.

$$JaccardSimilarity = \frac{x}{x+y+z} \tag{4.3}$$

Since the combination $(0,0)$ signifies the absence of elements in both sets of a characteristic matrix, it does not play any role in the Jaccard Similarity calculation. While the signature matrix holds signature values, there is no restriction on the combination of the pairs of a signature matrix to evaluate the Jaccard similarity. Here, Jaccard Similarity is the ratio of cardinality of the union of two sets to the cardinality of the intersection of two sets. To estimate similarities between users from the signature matrix can be calculated using Equation 4.4:

$$SIM(X,Y) = \frac{N_{mat}(X,Y)}{b} \tag{4.4}$$

Figure 4.1 shows the generated signature matrix using one permutation hashing. In the signature matrix rows correspond to the sets s_1, s_2, s_3 and s_4 and columns correspond to the buckets b_0, b_1 and b_2. Table 4.4 shows jaccard similarities between each pair of sets in the characteristic matrix as well as in the signature matrix.

[3]In the given example, $c = \lceil \frac{3}{2} \rceil + 1 = 3$

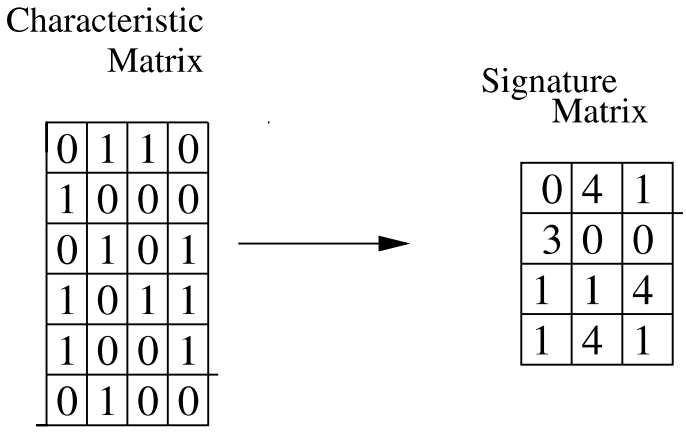


Figure 4.1: Signature matrix generation using one permutation hashing.

Table 4.4: Jaccard similarities.

Pair of sets	1-2	1-3	1-4	2-3	2-4	3-4
Characteristic matrix	0	0.25	0.5	0.25	0.2	0.25
Signature matrix	0	0	0.67	0	0	0.33

Even after significantly reducing the data, one permutation hashing preserves original jaccard similarities among all the pair of users.

User-user similarities are computed, and we get the similarity score among all individuals. After taking a threshold value, we determine the users that become the part of a set of candidate pair of users. It is easier to predict preferences for a user in a recommender system based on the similarity scores of the nearest neighbors as well.

4.1.2 Autoencoder

An autoencoder is an unsupervised deep neural network. The objective of the encoder model is to produce outputs identical to inputs [57, 73, 87, 121, 132]. The working principle of an autoencoder is straightforward. Figure 4.2 presents the basic encoder model. The input and hidden layer act as an encoder, while the hidden layer and output layer constitute a decoder. The fed inputs are encoded and decoded through hidden layers, and the output is matched to the input. We have some input x, somehow model reduces the dimensionality using an encoder; we encode the data into z (z is also known as a bottleneck which represents the compressed representation of our input data or some em-

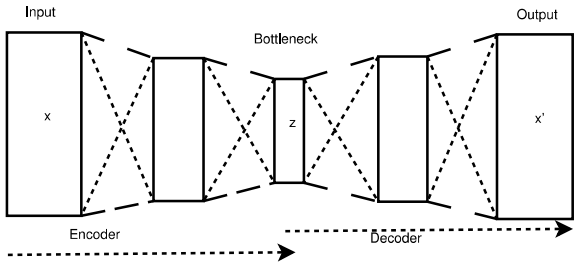


Figure 4.2: Encoder Model

bedding such that $dim(z) < dim(x)$, dim signifies dimension here. The bottleneck is the lower dimensional hidden layer where the encoding is produced) and then from this z, the model will be able to reconstruct our original input x or (x').

Mathematically, the encoder encodes the input data $x = \{x_1, x_2, ..., x_n\}$ which is a vector of dimension 'n' into a hidden input representation $z = \{z_1, z_2, .., z_m\}$ which is a vector of dimension 'm' by a function $Y = f(A) = AC(W.A + b)$ where $AC()$ is an activation function and W,b are connection weights and neuron bias, respectively.

The decoder maps the hidden representation back to a reconstruction $x' = \{x'_1, x'_2, ..., x'_n\}$ by a function $x' = g(z) = AC'(W'.z+b')$ where $AC'()$ is the decoder's activation function. The activation function is helpful in learning the non-linearity.

An autoencoder is trained to minimize the reconstruction error between A and A' . It is done by back-propagation using square error or cross-entropy metrics. An autoencoder can extract the non-linear features automatically. It can easily understand the user's expectations and properties of items in a recommender system. We have used this technique to explore users with similar characteristics. In the case of an autoencoder, we can model users, one user at a time, or we can model items, one item at a time (rows or columns). Missing entries are set to 0 in the input and not considered in the output.

There are other variants of autoencoders, such as viz. stacked autoencoder, sparse autoencoder, denoising autoencoder, and deep autoencoders [121]. A basic autoencoder framework has only one hidden layer. Stacked autoencoders hold multiple encoding or decoding layers.

4.1.3 OPHAencoder

OPHAencoder is the acronym for combining one permutation hashing (OPH) and autoencoder (Aencoder) techniques used for clustering. We generated the signature matrix

Table 4.5: Pairwise similarities among users from the signature matrix into a similarity matrix

1	0	0	0.67
0	1	0	0
0	0	1	0.33
0.67	0	0.33	1

Table 4.6: Similarity matrix

1	0	0	0.67	0.5	0
0	1	0	0	0	0.5
0	0	1	0.33	0	0
0.67	0	0.33	1	0.25	0
0.5	0	0	0.25	1	0.67
0	0.5	0	0	0.67	1

to find a similar pair of users using one permutation hashing. We construct a similarity matrix. Our objective is to explore a set of users who could be group members. It is observed that the pairwise similarity of a dissimilar pair in the characteristic matrix is mostly mapped to 0 in the signature matrix (Table 4.4). We represent the Jaccard Similarities among users in a similarity matrix. We consider the pairwise similarity threshold ≥ 0.5 to be a candidate member of a group. We do not consider the entry of 1 from this matrix in the calculation as it shows the same pair's similarity. In Table 4.5, only user pair 1-4 holds this property and forms a group of size two. Classical spectral clustering also provides the same result. Let us consider the case of a representation of pairwise similarities among users from a signature matrix into a similarity matrix of dimension 6×6 (Table 4.6).

Here, clustering based on threshold value will not be a proper choice. We applied spectral clustering [4] in the similarity matrix to form the groups. If we consider two clusters, users 2, 5 and 6 will become members of one cluster, whereas users 1, 3 and 4 will be in another cluster. This way, we can find similar users to form a cluster.

The clusters formed in the obtained signature matrix using traditional clustering approaches, viz., K-means, DBSCAN and spectral clustering are coarse-grained. So, we have also incorporated the autoencoder to find similar users. Applying a two-way filtering method to obtain a potential group from a given dataset becomes essential.

[4]https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html

In the case of an autoencoder, the utility matrix is sparse. The entries of '1' in the matrix indicate that the users have rated those products and vice versa. A neural network takes an individual user preference vector as input in an autoencoder. These inputs are fed into an input layer during the training phase. The input samples are compressed at the hidden layer, and the aim is to reconstruct a similar input at the output layer using this hidden layer. The activation functions in both the input and output layers remain the same. In the hidden layer, *tanh* activation function can be used. We train this unsupervised neural network, and the trained model can predict the rating. The built model can find unknown or missing ratings. In this case, predicting unknown ratings involves finding similar users also. In the collaborative neural network, we need to calculate the similar users' (k-nearest neighbors) similarity scores to predict a target user's rating. To learn the latent features to find the k-nearest neighbors using this technique is effective. These k-nearest neighbors are considered as a group of users who predict the same item. The union of these k-nearest neighbors comprises members of a group. The union comprises elements (users) which lie in these two different sets together. Finally, the union of the sets of users from both techniques will generate a final group.

4.2 Experimental setup and result analysis

We took two real-world datasets: Two variants of MovieLens dataset, i.e., MovieLens (ml-100k) and MovieLens (ml-1m) [5] and Amazon-music (Digital Music) [6] for the evaluation of the proposed work.

MovieLens(ml-100k): Movielens (ml-100k) dataset contains 943 distinct users and 1682 distinct items. Users have provided ratings over 100000 transactions in the rating scale of 1-5. It is observed that on an average at least twenty preferences are rated by each user in the ml-100k dataset.

MovieLens (ml-1m): MovieLens (ml-1m) dataset has 6040 distinct users and 3706 distinct items. The total number of instances are 1000209.

Amazon-music (Digital Music): The digital music dataset holds reviews and metadata information from amazon. The digital music dataset has 5148 distinct reviewers

[5] https://grouplens.org/datasets/movielens/

[6] https://jmcauley.ucsd.edu/data/amazon/

and 2582 distinct asin (a unique number assigned to each product/music). "asin" is an attribute of the dataset. "asin" corresponds to an item_id. The rating scale(named "overall") is 1-5. The total number of reviewers and asin interactions are 51796. Initially, all our experiments are conducted on a computer system with the following specifications:

- Processor: Intel(R)Core(TM) i7-7700 CPU @3.60GHZ
- RAM: 8 GB (7.88 GB usable)
- Disk: 1 TB
- System type: 64-bit operating system, x64-based processor
- Operating System: Windows 10

Later, some of the experiments are also conducted on a CPU node of the ParamShivay supercomputer. It has an IntelxeonSKLG-6148 processor and 192 GB RAM with Linux operating system installed. All the algorithms are implemented using python programming only.

4.2.1 Pre-processing step:

In the MovieLens (ml-100k & ml-1m) and the Amazon-music, we dropped the timestamp and title attributes from both the datasets, respectively, as it does not play any role in our model during the data pre-processing task. In a formed group, the preference vector of the user of a group contains movies having a rating higher than 2 (on a scale of 1-5) given by the user. We consider the order of items/asin as they appear in the datasets. In the case of an autoencoder model, files u1.base and u1.test of the ml-100k dataset are taken as training-set and test-set, respectively. In the neural collaborative filtering model, the number of latent factors is taken as 40.

4.2.2 Group formation step:

We blended one permutation hashing (OPH) and autoencoder to detect and compose groups automatically. Assume that the rating matrix (R) is a $m \times n$ size matrix containing "m" users and "n" items. We normalize the ratings to 0 and 1 for the Movielens and Amazon-music datasets. We do this to fit our experimental model in the case of OPH.

We generate a characteristic matrix by applying the transpose of the user-item matrix by which the collection of sets (users) can be visualized. We permute the rows using only one hash function on the characteristic matrix. We calculate the signatures of each attribute of the characteristic matrix using OPH and store it as the signature matrix. The signature matrix contains the same number of users/columns of the characteristic matrix and rows represent the buckets. In the case of the ml-100k dataset, the dimensions of the obtained signature are 43×943. The dimensions of the original characteristic matrix are 1682×943. In the case of ml-1m dataset, The dimensions of the original characteristic matrix are 3706×6040. In the case of the ml-1m dataset, the dimensions of the obtained signature are 62×6040. We compute the candidate members for the groups.

An autoencoder can extract the features and determine the similar kind of users. Similarly, the autoencoder provides the sets of users for the groups. In the autoencoder model, during model training as an optimiser parameter, the learning rate is taken as 0.01 and weight decay = 0.5. During training, the number of the epoch was set to 200. After 200 epochs, final loss and test loss were tensor(0.9523), tensor(0.9116), respectively. The candidate pair of users from both sets become members of the automatically identified group.

We applied traditional clustering approaches to measure the effectiveness of formed groups. Since we got a similar number of users in the automatically identified group, we conducted comparative studies among these clustering approaches only. The parameters of the Louvain clustering algorithm [12] for both the datasets are: we used three Principal Component Analysis (PCA) processing components, euclidean distance as distance measure approach with a k-nearest neighbor where k is 30 to form the clusters. For fuzzy c-means [16] technique, the euclidean distance is taken as similarity measure with k-nearest neighbors where c is eight. To generate the score vector [1], we took values of a and regularization factor c as 2 and 1, respectively.

4.2.3 Recommendation step:

Once the community (group) is formed, we have to recommend items that can provide maximum satisfaction to the group members. To generate the recommendation list for a detected group, we are using consensus functions, which also determine that a user of a group is maximally satisfied with the recommended item-set. This system recommends

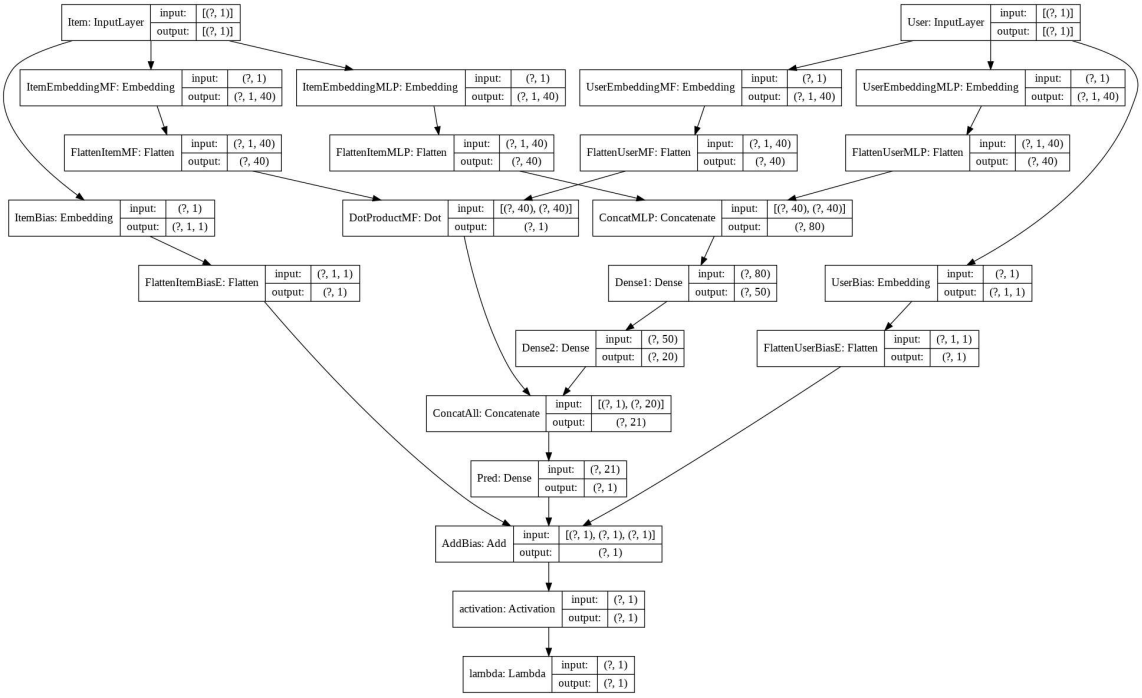


Figure 4.3: NCF model summary on the formed group by OPHAencoder

Table 4.7: Recommendation result on the formed group using NCF model

MODEL	RMSE Score	NDCG@k	Precision@k	Recall@k
NCF	1.058051	0.03412	0.024691	0.007816

items for a group by considering order preferences provided by the members of the group as well as assuming flexibility in the users' preference list. We made the prediction of the ratings of the formed group using neural collaborative filtering. A neural collaborative filtering (NCF) is a deep neural network that models the user and item interactions in the latent space effectively. NCF is a generalization of matrix factorization [100]. Figure 4.3 shows the layer-wise architecture of the neural collaborative filtering model in the formed group. The result of recommendations using neural collaborative filtering model of the formed group using OPHAencoder is presented in Table 4.7.

4.2.4 Baseline and state-of-the-art models:

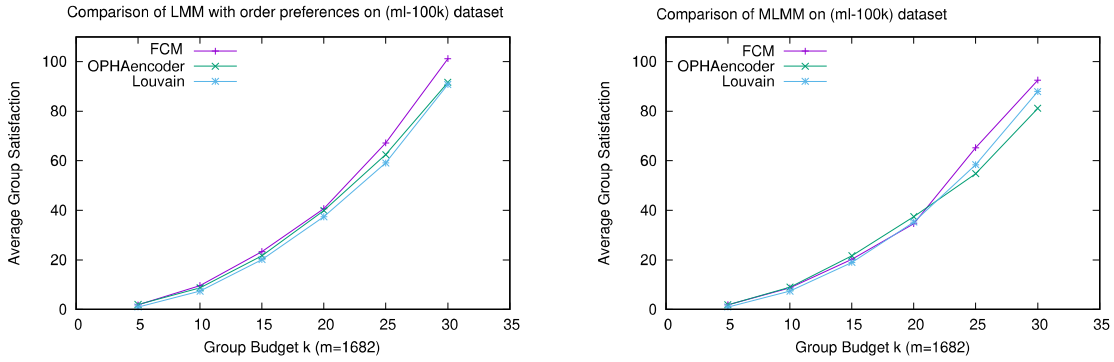
In the proposed work to evaluate the efficacy of the models and the comparison purposes, various state-of-the-art models have been taken.

Fuzzy c-means (FCM) [7]: Fuzzy c-means is a type of soft-clustering algorithm. Data points belong to other clusters in this clustering approach. Consequently, it is essential to assign weights to each data point and each group that indicates the degree to which the data belongs. Weights are chosen randomly, and there is a constraint that the sum of weights associated with each data point should be equal to 1. When using FCM in the given datasets, data points are not partitioned into the well separated clusters. We assigned weights to each data point and each cluster represent the degree to which the data belong to the cluster. Objective is the random weight assigned with any data point must be summed to 1.
Louvain clustering [12]: A community detection based clustering method that works on the principle of level-wise modularity optimization. The modularity function is used to calculate the modularity gain value of each vertex. In each iteration, vertices and edges are interconnected based on the modularity gain value of the nodes. The modularity gains of nodes are repeatedly updated in the obtained communities until it converges. Finally, we explore different groups.

K-means [16] is the most popular clustering method for the unsupervised task. The objective of the algorithm is to organize similar items into a group. Data are divided into multiple clusters. Matching items are placed in the same cluster to minimise intra-cluster difference, and the inter-cluster difference is maximised. Agglomerative clustering [20, 83] is a hierarchical clustering approach that groups similar objects into a group.

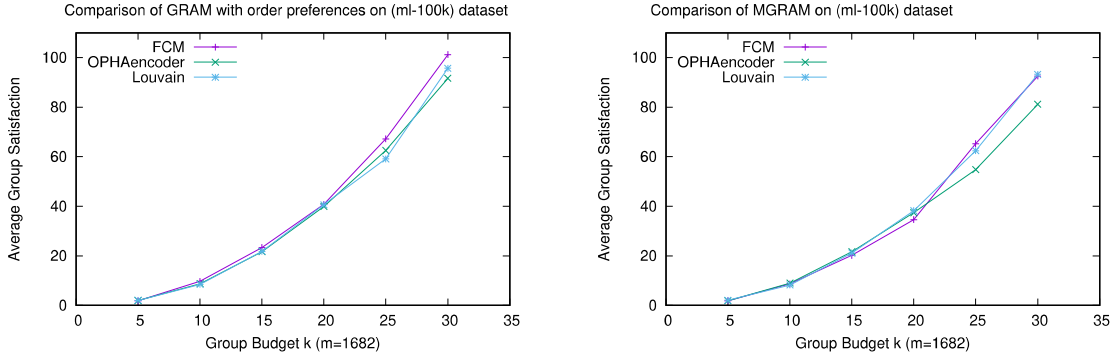
4.2.5 Results and analysis

We compare the result with other classical clustering approaches and a community detection approach to find the effectiveness of automatically detected groups using the proposed approach and depict the results. To prove the efficiency of the proposed system, we used the adopted versions of two widely known versions of consensus functions for recommendation: aggregated voting (AV) and least misery. The adopted versions that we used in this chapter are the least misery method with priority (LMMP), and modified least misery method with priority (MLMMP). Even though the greedy aggregated method (GRAM) and modified greedy aggregated method (MGRAM) aim to choose the best item greedily from the user satisfaction score matrix, it is still far from providing an optimal solution. Hungarian aggregated method (HAM) and modified hungarian aggregated method



(a) (b)

Figure 4.4: Average group satisfaction of cluster size ($70 \leq clusterSize \leq 80$) using a) LMM and b) MLMM on ml-100k dataset.

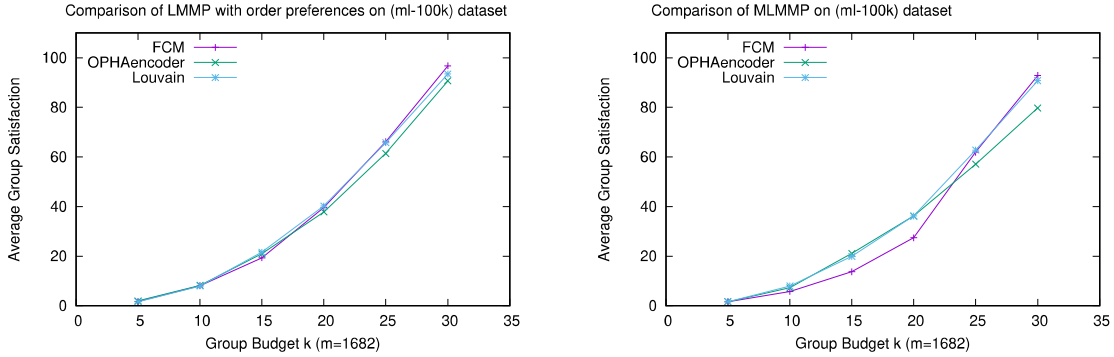


(a) (b)

Figure 4.5: Average group satisfaction of cluster size ($70 \leq clusterSize \leq 80$) using a) GRAM and b) MGRAM on ml-100k dataset.

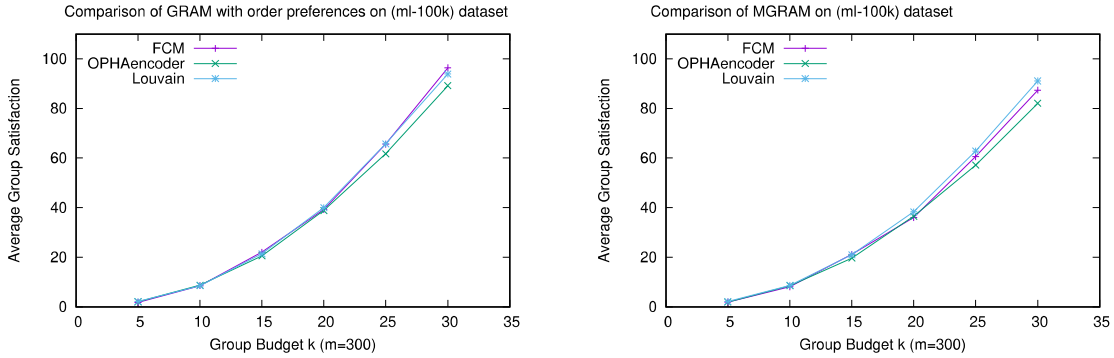
(MHAM) offer an optimal solution.

In the NCF model, during training a model, hyper-parameters viz. batch-size and epochs were taken 128 and 5, respectively. The validation set took 20% part. After five epochs, the final loss and validation loss are 0.6692 and 1.1780, respectively. Figure 4.8 shows the evaluated result of NCF model. The recommendation by the whole group on the NCF model using aggregated strategy is item 39. The obtained outcome is satisfactory. We varied the parameters of group budget (k), item (m) and numbers of the users of the group (n) to check the efficacy of the proposed method on order preferences and the flexible-size preferences-based model. The order preference model provides a set of items that users would like to see in the generated recommendation list along with an order



(a) (b)

Figure 4.6: Average group satisfaction of cluster size ($70 \leq clusterSize \leq 80$) using a) LMMP and b) MLMMP on ml-100k dataset.



(a) (b)

Figure 4.7: Average group satisfaction of cluster size ($70 \leq clusterSize \leq 80$) using a) GRAM and b) MGRAM when item (m)=300 on ml-100k dataset.

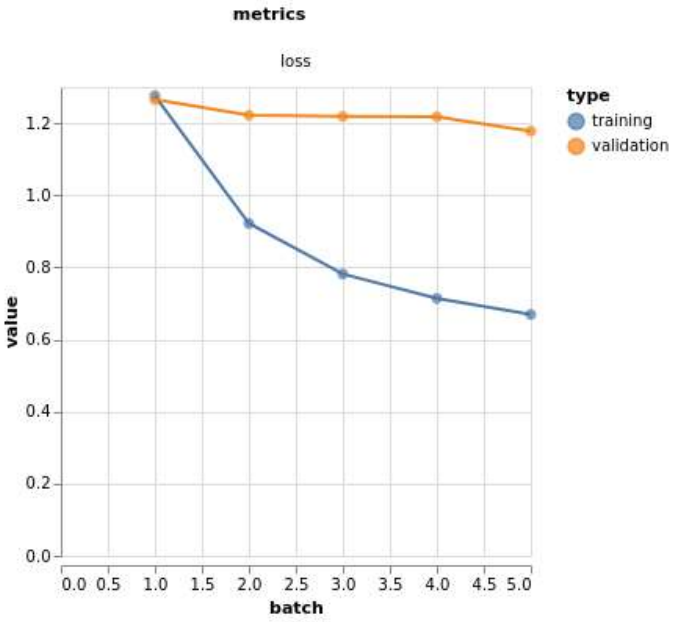
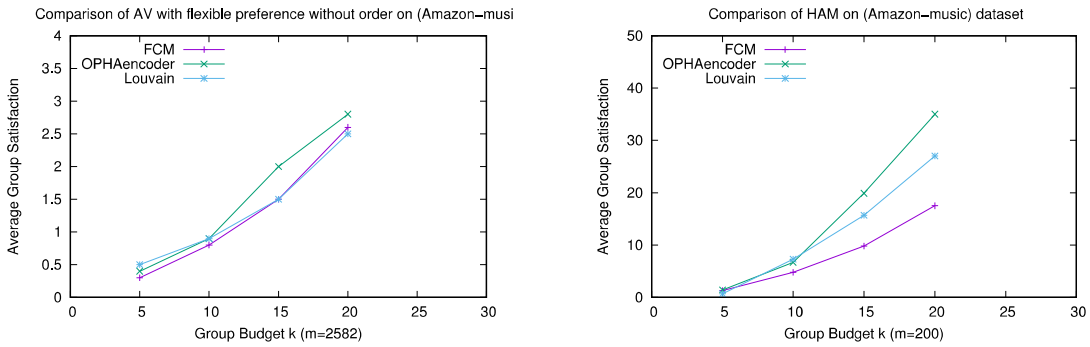


Figure 4.8: Plotted curve (value Vs. batch) for both training and validation sets of NCF model

in which a user wants those items to appear [1]. In the available literature, none of the group satisfaction measures considers the order of preferences; therefore, user satisfaction with order (USO) [1] is also proposed, and an evaluation metric called group satisfaction score metric is introduced. The average group satisfaction score signifies the ratio of total group satisfaction score to the number of users in a particular group.

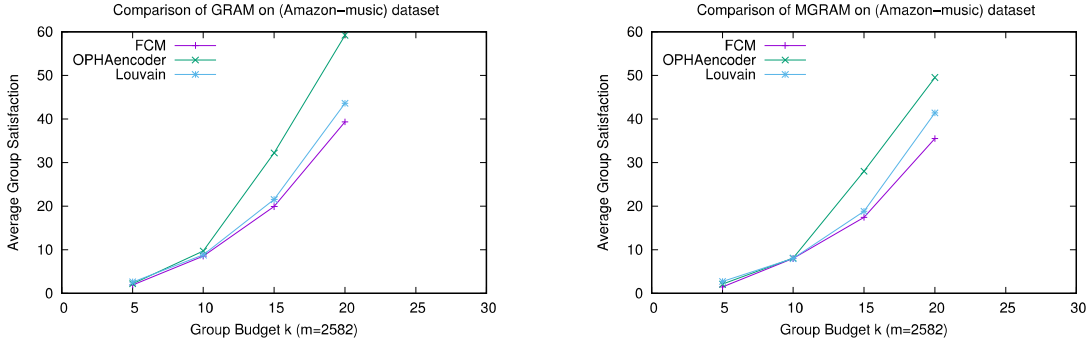
It holds similar characteristics to results depicted in baseline models [1, 35], i.e., the overall group satisfaction or average group satisfaction increases by increasing the m, k and n of an auto-detected group. Figures 4.4, 4.5, 4.6, 4.7, 4.9, 4.10, 4.11 and 4.12 validate these characteristics. The total group satisfaction or average group satisfaction is less when considering flexible preferences without order. In contrast, order preferences provide a better result. Figures 4.4, 4.5, 4.6, 4.7, 4.9, 4.10, 4.11 and 4.12 show that the proposed approach is a promising approach for auto-detecting a group and improves the overall group satisfaction score. We conclude from these figures that the proposed method achieved an outcome similar to established clustering approaches, and using this method to auto-detect groups in the group recommendation is promising and overwhelming.

Table 4.8 and Table 4.9 provide a comparison based on the total group satisfaction score of the automatically detected group of different sizes (n) using GRAM and LMM



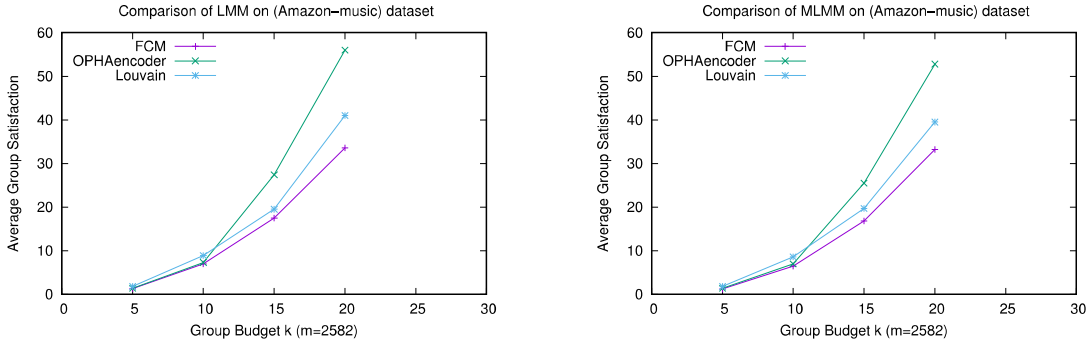
(a) (b)

Figure 4.9: Average group satisfaction of cluster size ($50 \leq clusterSize \leq 55$) using a) AV with flexible preference without order and b) HAM on Amazon-music dataset.



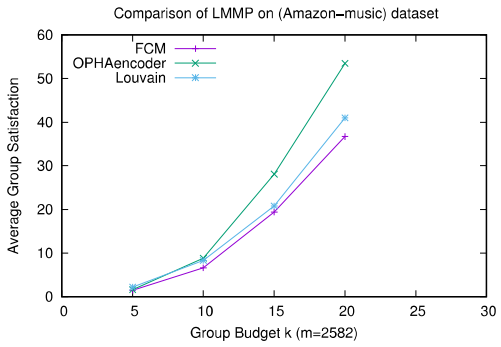
(a) (b)

Figure 4.10: Average group satisfaction of cluster size ($50 \leq clusterSize \leq 55$) using a) GRAM and b) MGRAM on Amazon-music dataset.

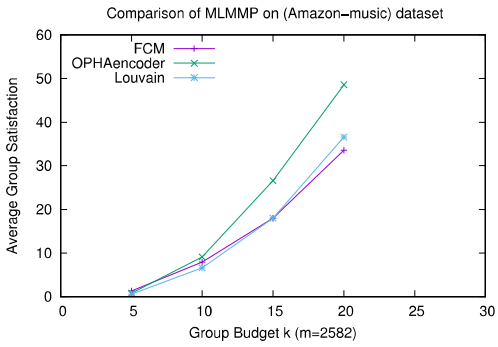


(a) (b)

Figure 4.11: Average group satisfaction of cluster size ($50 \leq clusterSize \leq 55$) using a) LMM and b) MLMM on Amazon-music dataset.



(q)



(r)

Figure 4.12: Average group satisfaction of cluster size ($50 \leq clusterSize \leq 55$) using a) LMMP and b) MLMMP on Amazon-music dataset.

Table 4.8: Comparison with baseline methods on the basis of total group satisfaction scores of the automatically identified group of different group sizes (n) and (m=3706) using GRAM on MovieLens (ml-1m) dataset

k	Users(n)	OPHAencoder	K-means	Agglomerative	Louvain	FCM
5	45	334.9	249.9	287.3	256.8	318
10	44	1008.1	764.5	813.7	741.7	983.2
15	37	1605.1	1467.3	1482.9	1227.8	1713.9
20	30	2134	2391	2282.6	1870.2	2408.3
25	28	2911.4	3468.2	3376.1	2852.7	3553.4

algorithms on the MovieLens (ml-1m) dataset. We composed and detected groups of the different number of users who opted for at least 'k' items and accordingly evaluated the overall group satisfaction scores. The results in the tables have been depicted and concluded that the proposed approach is a promising method to auto-detect the groups.

Time complexity:

We calculate the time complexity of the OPHAencoder algorithm. We check the efficacy of this algorithm by determining how fast this algorithm runs as the input size increases. In the OPHAencoder technique, the input length is $m \times n$ where n represents the number of users and m is the total number of items. The OPHAencoder model unifies two different techniques, i.e., stacked autoencoder and one permutation hashing. In the worst-case scenarios, One permutation hashing requires applying $l = 1$ number of the hash function. The worst-case time complexity to compose a group using this algorithm is $\mathcal{O}(mn + n^2)$.

Table 4.9: Comparison with baseline methods on the basis of total group satisfaction scores of the automatically identified group of different group sizes (n) and (m=3706) using LMM on MovieLens (ml-1m) dataset

k	Users(n)	OPHAencoder	K-means	Agglomerative	Louvain	FCM
5	45	229.9	201.9	222.1	155.7	188.9
10	44	910	642.8	697.8	634.9	931.3
15	37	1483	1213.1	1245.5	1110.7	1653.9
20	30	1747.9	1963.5	1886.7	1779.1	2345
25	28	2537.7	2897.1	2925.6	2689.5	3367.6

The worst-case time complexity to form a group using a stacked autoencoder algorithm is $\mathcal{O}(NE \times k)$. Where N is the number of training examples, and E represents the number of epochs. k is the total time required to complete one epoch by an autoencoder model. Autoencoder can be evaluated on multiple model architecture. Autoencoder model also train on GPU. GPU works parallelly and provides a more optimize way to train a model. In this scenario, determining the time complexity of the stacked autoencoder will be reduced. In the worst-case scenario, the time complexity of the OPHAencoder model to auto-detect a group is $\mathcal{O}(mn + n^2 + NE \times k)$.

4.3 Summary

We introduced a novel approach to explore communities (groups) with similar properties and be part of an automatically identified group. We investigate the superiority of the formed groups in fixed and variable-size preferences based models. It also observes that the members of the groups participate in a decision-making process to achieve consensus profoundly in the established group. We form groups by dealing with the curse of dimensionality problem. Compared to other baseline clustering approaches, the proposed approach mitigates the information overloading problem. The obtained results show that the proposed method helps to detect and compose a group automatically in group recommendations. Systems also predict the preferences of the user of the group, and the results are encouraging.