Chapter 3

Automatically detecting groups using locality-sensitive hashing in group recommendations

This chapter is based on our journal article published in [70]. This chapter focuses on "automatically detected groups" formation for order and flexible preferences in group recommendation using locality-sensitive hashing. The MinHash technique is applied on a characteristic matrix to generate the signature matrix. The signature matrix is the reduced representation of the characteristic matrix and preserves the Jaccard Similarity to a great extent. Locality-sensitive hashing is applied on the signature matrix to determine similar users efficiently. Similar users will be the members of an automatically identified group. Therefore, the group members are maximally satisfied with recommended items. This work also studies the performance of benchmark clustering approaches in group formation. We experimented on real-world datasets and found that the proposed models to identify communities in group recommendation maximizes consensus among users in a group.

Outline: The rest of the chapter is organized as follows. Section 3.1 presents problem definition and the overview of the proposed model. Section 3.2 discusses the methodology of the proposed model. Experimental setup and results are discussed in Section 6.2. Finally, Section 5.3 summarizes this work.

3.1 Problem definition and model overview

Let a set of users, set of items and rating scale are denoted as U, I and r respectively, where

$U = \{u_1, u_2, u_3, ..., u_m\}$

$I = \{i_1, i_2, ..., i_n\}$

$r = [1, 5]$ or r={Like,Dislike}

Here, individual user u_i gives a rating to an item u_j using r. The rating matrix R is $m \times n$ matrix containing "m" users and "n" items. If a user i rates an item j, then the corresponding cell R_{ij} in this matrix holds a rating value. The provided rating scale varies for different datasets. The explicitly provided ratings of a utility matrix need to be converted into "1" and "0" where "1" specifies like, and "0" represents dislike. All the elements of the characteristic matrix will be binary after normalization. Binary ratings feed into the characteristic matrix as an input to the model.

Further, the characteristic matrix condense into the signature matrix of size $l \times m$ using the MinHash technique, where l is the total number of hash functions applied to reorder the items. The Jaccard Similarity between a set of users U preserved in this representation aims to find candidate pairs of users who are similar using this signature matrix. The computed signature matrix uses the LSH technique to cluster the users and auto-detect a group. Let G_p be the group of users where $G_p \subseteq U$ such that group members have a higher similarity. After aggregating the individual preferences of users of the auto-detected group (G_p), the goal of the model is to recommend itemset of size k (group budget) with a more group satisfaction score when incorporating order or flexibility in user preferences. The flexibility in user preferences refers to the variable length of user preferences. The flexibility in the preference list (pl) of a user means the size of pl (S_{pl}) can be up to group budget (k), i.e., $\forall pl \in G_p, S_{pl} \leq k$ where $k > 0$.

3.2 Proposed Model

3.2.1 MinHash technique

MinHash played a pivotal role in many applications, and it seems promising to use this approach in group formation task. MinHash is a fast similarity search technique. Jaccard

Similarity [5] is a similarity measure in the context of MinHash. During figuring out duplicate documentation, it reduces large sets of unique shingles (k-gram) into much smaller representations called "signatures". Further, these signatures help to calculate estimated Jaccard Similarity, which is a good approximation of the results of Jaccard Similarity on the original characteristic matrix. The signature matrix is a smaller representation of the original characteristic matrix and preserves similarities to a great extent.

In 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 the utility matrix. Since users provide ratings explicitly in some rating scale, it is required to convert these ratings into '0' or '1'. In the case of ml-100k dataset; the rating scale is from 1 to 5. To normalize this rating to '0' or '1'. We divide each rating score with (maximum rating score+1) i.e; (5+1=6). We get the real values between '0' and '1' and '0.5' is taken as the threshold. So, a value greater than or equal to the threshold becomes '1' and if it is less than the threshold, it becomes '0'.

In the case of Last.fm dataset, user-artist pair provides user preference vectors having their preferences of artists and their associated weights. Here, artists are analogous to items. So, we calculate the cosine similarity score between items/artists in both the datasets separately to normalize those ratings/weight-scores. If we consider ml-100k dataset; we calculate the item-item similarity using cosine similarity from user rating matrix. A smaller cosine similarity value tends to zero while a greater cosine score tends to one after fixing at a reasonable threshold. Thus, representing a user-item rating matrix in the form of $(0, 1)$-matrix is considered as an input of a characteristic matrix. To justify this representation, we use KNN-item-based collaborative filtering to predict the 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 ml-100k dataset, using original utility matrix; KNN item-based collaborative filtering gives Mean RMSE score as 0.99. RMSE takes two vectors as input-original movie ratings and predicted movie ratings. In 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 the predicted ratings for the items by a user. 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 neighbors of the items. 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:

$$P_{u,i} = \frac{\sum_{\forall_j \in z_u(i)} (JS_{i,j} \times R_{u,j})}{\sum_{\forall_j \in z_u(i)} |(JS_{i,j})|} \tag{3.1}$$

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 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 RMSE function. We got a similar performance, and there is no significant deviation in the outcome. While, considering the normalized ratings in $(0, 1)$-matrix, Item-based collaborative filtering gave RMSE score of 0.37. We have to compromise on accuracy as there is no well-established encoding standard for the task. After this, MinHash technique is applied to generate signature matrix. Figure 3.1 depicts its details more elaborately. Figure 3.1 shows the signature matrix generation using the MinHash technique. In the MinHash technique, while generating randomly permuted rows, it may not be feasible to generate a random number for a large dataset. So, it is preferable to use a hash function to map elements of the sets to integers. Hash function should be in the form of: $h(x) = (px + q) \text{mod} m$ where:

1. x is the row number of original characteristic matrix i.e. $x \in 0, 1, 2, ..., n$.

2. p and q are any random numbers smaller or equal to the largest row number and both must be unique for each signature.

3. m is a prime number larger than the largest row number (n).

A column of a characteristic matrix is considered as a set and we have four sets (s_1, s_2, s_3, s_4) in characteristic matrix as given in Figure 3.1. Table 3.1 represents three permuted columns (P_1, P_2, P_3) independent of each other using three different hash functions. If, for any hash value, the corresponding row of a characteristic matrix contains 1, then that hash value will be MinHash signature of that corresponding set. Once we

Table 3.1: Permuted rows using hash functions for characteristic matrix.

Row	s_1	s_2	s_3	s_4	$P_1((x+2)mod5)$	$P_2((2x+3)mod5)$	$P_3((3x+4)mod5)$
0	1	0	1	1	2	3	4
1	1	0	0	0	3	0	2
2	0	1	1	1	4	2	0
3	0	1	0	1	0	4	3
4	1	0	1	1	1	1	1

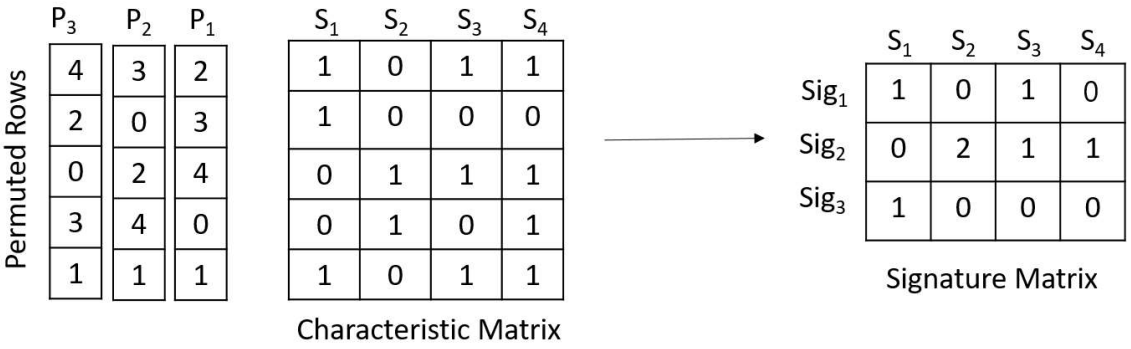


Figure 3.1: Signature matrix generation using MinHash.

Table 3.2: Jaccard similarities.

Pair of sets	1-2	1-3	1-4	2-3	2-4	3-4
Characteristic matrix	0	0.5	0.4	0.25	0.5	0.75
Signature matrix	0	0.33	0	0	0.67	0.33

get MinHash signature of a particular set for an individual permuted column, we will not consider it for the next hash value. We go for hash values in ascending order, i.e., min-wise permuted row order. A permuted column is used to compute a signature row of the signature matrix.

For the first signature row, initially, we take 0 from P_1, and MinHash signatures of sets S_2 and S_4 are set to 0 as the corresponding row of these two sets have 1 in the characteristic matrix. In next round, hash value 1 from P_1 is selected and MinHash signature of sets S_1 and S_3 are set to 1. We computed MinHash signatures for all the sets of P_1, and the signature (Sig_1) in the signature matrix is (1,0,1,0). After computing final signature matrix by applying the same process, the Jaccard Similarity is computed on this similarity matrix. 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, $|Type(0,0)|= d$.

$$JaccardSimilarity = \frac{x}{x+y+z} \tag{3.2}$$

Since the combination $(0,0)$ signifies the absence of elements in both the sets of characteristic matrix, it does not play any role in Jaccard Similarity calculation. While signature matrix holds signature values, there is no restriction on the combination of the pairs of 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. Table 3.2 shows Jaccard Similarities between each pair of sets in the characteristic matrix as well as in the signature matrix. Even after significantly reducing the data, MinHash preserves original Jaccard Similarities.

Groups are categorized into homogeneous and heterogeneous groups based on their composition. A homogeneous group consists of the members having similar interests, whereas heterogeneous group members consist of diverse interests. Our work deals with the formation of automatically identified groups using locality-sensitive hashing.

3.2.2 Locality-sensitive hashing for finding the candidate pairs of users

Locality-Sensitive Hashing (LSH) [54] is an efficient approach to identify approximate nearest neighbors [4]. LSH is based on the principle that if there are two feature spaces

close to each other, they are likely to have the same hash (reduced representation of data). It preserves local relations of data while significantly reducing the dimensionality of the dataset. In MinHashing; the similarity between each pair of users can be estimated by monitoring relevant users for each item in a MinHash index while generating the signature, and it produces a smaller representation of the original matrix and preserves the Jaccard Similarity between all the pairs of users. Finding similar pairs of users in the computed signature matrix is computationally expensive because there can be too many pairs of users. Therefore, we incorporate locality-sensitive hashing to find candidate pairs of users to cluster them into a particular group. Any pair that is hashed to the same bucket for at least one hash function is known as a candidate pair.

The signature matrix is divided into b bands of r rows each. For a particular band, a hash function is applied to the vectors of r rows and hash them into buckets. The same hash function is applied for every band, but each band has a separate set of buckets. Similar users are likely to be hashed into the same bucket than dissimilar ones [6, 44]. Similar users hashed into the same bucket are considered as candidate pairs (Figure 3.2).

Dissimilar pairs that are hashed to the same bucket are false positive, and we expect very few dissimilar pairs. Similar pairs are considered as false negative if they do not hash to the same bucket for any hash function. If s be the Jaccard Similarity for two items, the probability that the MinHash signature agrees in all of the rows of at least one band for these two items is $1 - (1 - s^r)^b$, and this probability determines the candidate pairs. We take a threshold value and calculate Jaccard Similarity for each candidate pair. The candidate pairs that have Jaccard Similarity above the threshold are the desired similar users.

However, we are limited by probability as it does not provide strong evidence that the users are alike, so we need to perform amplification to reduce the trade-off between false negative and false positive.

Amplifying a locality-sensitive family:

In the MinHash technique, jaccard distance is used to measure the distance between sets where user preferences are represented as sets. In MinHash function (mh) two sets X and Y can be a candidate pair if and only if $mh(X) = mh(Y)$. Suppose d_1 and d_2 be two

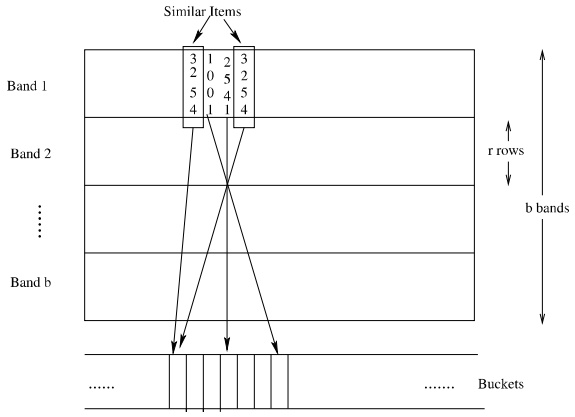


Figure 3.2: Applying LSH technique in generated MinHash Signature to find candidate pairs of users.

distances according to the distance measure d[1] such that $0 \leq d_1 \leq d_2 \leq 1$. p_1 and p_2 are probabilities where $p_1 = 1 - d_1$ and $p_2 = 1 - d_2$. A family F of functions is said to be (d_1, d_2, p_1, p_2)-sensitive family[2] for every $f \in F$, where:

1. If $d(X, Y) \leq d_1$, then the probability that $f(X) = f(Y)$ is at least p_1 .
2. If $d(X, Y) \geq d_2$, then the probability that $f(X) = f(Y)$ is at most p_2 .

Suppose we are given a (d_1, d_2, p_1, p_2)-sensitive family, then:

1. if $f_1, f_2...f_n$ are (d_1, d_2, p_1, p_2)-sensitives then we can concatenate them together to construct a (d_1, d_2, p_1^n, p_2^n) sensitive hash to reduce the probability of being irrelevant.
2. if $f_1, f_2...f_n$ are (d_1, d_2, p_1, p_2)-sensitives then we can perform union of all of them to construct a $(d_1, d_2, 1 - (1 - p_1)^n, 1 - (1 - p_2)^n)$ sensitive hash to increase the probability of being relevant.

The above combination of concatenation and union gives probabilities approximately close to 1.

Advantages of the proposed method

In the existing literature (Chapter 2), traditional clustering approaches have been applied only to auto-detect the group in group recommendations. Using these clustering

[1]In this case d is Jaccard Similarity.

[2]If $d_1 = 0.3$ and $d_2 = 0.6$, family of minhash function is a(0.3,0.6,0.7,0.4)-sensitive family.

approaches, the time complexity of finding similar users using a characteristic matrix is quadratic. In this chapter, we propose to reduce the time complexity by considering a locality-sensitive hashing technique. The MinHash technique represents the original characteristic matrix in a reduced representation (a signature matrix) and preserves the original Jaccard Similarity scores. Further, we consider the reduced dimensions (signature matrix) to find the candidate pairs of users. So, the dimensions are also reduced in this approach. In addition, an auto-detected group using this approach is very promising.

Weight updation of each member of an automatically identified group during recommendations

A key point in the group recommendation problem is the weight of each group member. In the case of order or flexible-size user preferences to accurately obtain the weight of each group member solely depends upon values of score vector [1].

Example 2: There are *n(=7)* users of an automatically identified group and *m(=8)* items with the group budget of *k(=5)*. Flexibility in each user preferences are given as follows:

{2, 6, 7}

{2, 3, 4, 6, 7}

{3, 8}

{1, 2, 6, 8}

{5, 7}

{1, 3, 4, 5, 8}

{7}

User preference matrix U is of dimension $k \times m$, where m is the total number of items. Each item has a unique *item_id* that lies between 1 to m. Initially, all the entries of the user preference matrix are 0. In case of flexibility in user preferences, a user has a preference vector of length p, has a preference at position i as the data *item_id* j. The entry at $U_{i,j}$ is incremented by a factor of $\frac{k}{p}$. Table 3.5 provides the score vector values for group budget of size ($k = 5$). A score vector (s) contains the score to be added to the total group satisfaction when an item is placed with an absolute difference of i positions in the recommended list from the user preferred position of the preference vector. User

Table 3.3: User Preference Matrix

Position	Items							
	1	2	3	4	5	6	7	8
1	2.25	2.67	2.5	0	2.5	0	5	0
2	0	1.25	2	0	0	1.67	2.5	2.5
3	0	0	0	2	0	1.25	1.67	0
4	0	0	0	0	1	1	0	1.25
5	0	0	0	0	0	0	1	1

Table 3.4: User Satisfaction Matrix.

Position	Items							
	1	2	3	4	5	6	7	8
1	13.5	21.67	24.07	7.77	18.5	15.92	51.06	18.95
2	10.2	19.59	23.34	9.07	15.23	19.56	48.74	23.36
3	8.75	16.03	18.79	12	14.26	19.6	44.66	20.9
4	7.88	14.19	16.52	9.07	14.75	18.15	39.32	21.76
5	7.28	13.01	15.09	7.77	12.63	15.23	37.41	20.42

Table 3.5: score vector (s) when $k = 5$.

Index	0	1	2	3	4
score	6	4.54	3.89	3.5	3.24

preference matrix and score vector are used to determine the values of user satisfaction matrix S.

Table 3.3 consists of real values which are basically weights of entries of all the users by placing an item j at position i. User satisfaction matrix $S_{i,j}$ denotes the group satisfaction score matrix when an item j is placed at position i.

$$S_{i,j} = \sum_{p=1}^{k} s_{|p-i|} \times U_{p,j} \tag{3.3}$$

So, for the example under consideration: $S_{1,2} = s_0 \times U_{1,2} + s_1 \times U_{2,2} + s_2 \times U_{3,2} + s_3 \times U_{4,2} + s_4 \times U_{5,2} = 6 \times 2.67 + 4.54 \times 1.25 + 3.89 \times 0 + 3.5 \times 0 + 3.24 \times 0 = 21.67$ is the group satisfaction score when we placed an item 2 at position 1. Let us consider a GReedy Aggregated Method (GRAM) to produce a recommendation on the automatically identified group using the user satisfaction matrix (Table 3.4). The GRAM greedily chooses an item in each iteration from the user satisfaction matrix and adds that item to a recommendation list. In the above *Example 2*, item 7 at position 1 gives the maximum group satisfaction based on the user satisfaction matrix in the first iteration. Once we add this item to the recommendation list, all the entries correspond to the item's column index, and the row index of the position of the user satisfaction matrix becomes zero. In further iterations, these entries do not take part in the consideration. In the second iteration, item 8 is added for the second position in the recommendation list. We perform this iteratively and continue until the 'k' recommended list of items are generated. This method generates [7 8 6 3 2] with overall group satisfaction=**67.4** and average group satisfaction=**9.63**. Initially, the weight of each user is considered zero. Once an item is added to the recommendation list, the group members' weight gets updated according to the score vector(s) values. Accordingly, we update the weights of the users in further iterations until we get the recommendation list of size k. Table 3.6 shows weight updation of every member of the group. Alternatively, if we sum all the weights of users at the end of iteration 5, this is also equal to the total group satisfaction.

Table 3.6: Weight updation of each member of the group.

Iteration 1							
User	1	2	3	4	5	6	7
Weight	0+s[2]=3.89	0+s[4]=3.24	0	0	0+s[1]=4.54	0	0+s[0]= 6
Iteration 2							
User	1	2	3	4	5	6	7
Weight	3.89	3.24	0+s[0]=6	0+s[2]=3.89	4.54	0+s[3]=3.5	6
Iteration 3							
User	1	2	3	4	5	6	7
Weight	3.89+s[1]=8.43	3.24+s[1]=7.78	6	3.89+s[0]=9.89	4.54	3.5	6
Iteration 4							
User	1	2	3	4	5	6	7
Weight	8.43	7.78+s[2]=11.67	6+s[3]=9.5	9.89	4.54	3.5+s[2]=7.39	6
Iteration 5							
User	1	2	3	4	5	6	7
Weight	8.43+s[4]=11.67	11.67+s[4]=14.91	9.5	9.89+s[3]=13.39	4.54	7.39	6

3.3 Experimental setup and Result analysis

We took real-world datasets MovieLens (ml-100k) [3], and Last.fm[4] for the evaluation of the proposed model.

MovieLens(ml-100k): MovieLens (ml-100k) dataset contains 943 userIDs and 1682 movieIDs over 100000 ratings, and this dataset is 93.69% sparse, which means that the users have rated only 4.25% of the movies. This file contains four attributes: UserID, MovieID, Rating, and Timestamp. Here, UserID is the unique user ID in a range between 1 and 943, MovieID is the ID of the movie in a range between 1 and 1682, and Rating contains values on a scale of 1 to 5.

Last.fm: In this dataset, we have a listening history of users. There were three attributes in the rating file: User, Artists, and Weight. This dataset contains 92834 user-artists pair of 1892 users and 17632 artists.

3.3.1 Experimental setup and data preparation

Pre-processing step:

In ml-100k, during the data preprocessing task, we dropped timestamp attribute from the dataset as it does not play any role in our model. In a formed group, the preference

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

[4]https://www.last.fm/

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. In the case of Last.fm dataset, We have 92834 user-artist pair that provides 1892 user preference vectors with their artists' preferences. The order of the items (movies/artists) for both datasets is the same as they appear in their respective datasets.

Group formation step:

We use LSH 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 dataset. In the case of Last.fm dataset, the weight attribute contains 5436 different weights; we normalize these weights to the scale (0,1). We do this to fit our experimental model. We create a characteristic matrix by applying transpose in the user-item matrix by which the collection of sets (users) can be visualized. It requires to permute the rows of the characteristic matrix using 100 different hash functions. The system calculates the MinHash signatures of each characteristic matrix attribute and stores them in the signature matrix. The signature matrix contains the same number of users/columns as the characteristic matrix, but the rows equal the number of hash functions. Further, it requires applying the LSH technique on the signature matrix to find the candidate pairs of users. The candidate pairs of users become group members after amplifying the locality-sensitive hash.

To measure the effectiveness of formed groups, we applied classical clustering approaches and a community detection algorithm (Louvain clustering algorithm) to auto-detect the group. Louvain algorithm [13] works on a principle of level-wise modularity optimization. The proposed modularity function is used to calculate the modularity gain value of each node. In each iteration, nodes and edges are interconnected based on the modularity gain value of the nodes. The modularity gains of nodes are repeatedly updated in the communities till they converge. Finally, we explore different communities for a community detection task. Time complexity is the square of the network size. K-means [16] is a traditional and popular clustering method. The goal of K-means clustering is to organize similar items into a group. Data are divided into clusters. Similar items are placed in the same cluster in order to hold the properties that intra-cluster differences are minimized, and the inter-cluster difference is maximized. Agglomerative clustering [83]

is a hierarchical clustering approach that groups similar objects into a particular cluster. Since we got an equal number of users in the automatically identified group, we only conducted comparative studies using these clustering approaches. In the comparative study, we consider those clustering approaches which is used in auto-detecting groups in existing articles. We consider some other traditional clustering approaches also. In the case of DBSCAN, the obtained clusters did not provide a similar number of users corresponding to other approaches in comparative study. Also, some other clustering approaches do not provide it. So, we do not take into account those clustering approaches.

The parameters of the Louvain clustering algorithm [13] for both the datasets are: Model uses 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 K-means [16] and agglomerative clustering [83] technique, the euclidean distance is taken as a similarity measure with k-nearest neighbors where k is ten. To generate the score vector, we took values of a and regularization factor c as 2 and 1, respectively.

Recommendation step:

Once the community is formed, models recommend items to maximize group members' satisfaction. The model employs consensus functions to generate the recommendation list for a detected group, determining that a group user is maximally satisfied with the recommended item set. This system recommends items for a group by considering order preferences provided by the members of the group as well as assuming flexibility in the order of items in the users' preference list, i.e., when an order of items is essential and when an order does not matter.

3.3.2 Results and analysis

We compare results with other classical clustering and community detection approaches to determine the effectiveness of automatically detected groups using the proposed method and depict the results. To prove the efficiency of the proposed system, we used the adopted versions of two widely known consensus functions for recommendation: aggregated voting and least misery. The adopted versions that we used in this chapter are the Hungarian Aggregated Method (HAM), Modified Hungarian Aggregated Method (MHAM), 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 (MHAM) provide an optimal solution. We also consider most pleasure with priority (MPP) and most pleasure method with priority (MPMP) consensus functions to check the efficacy of the proposed models.

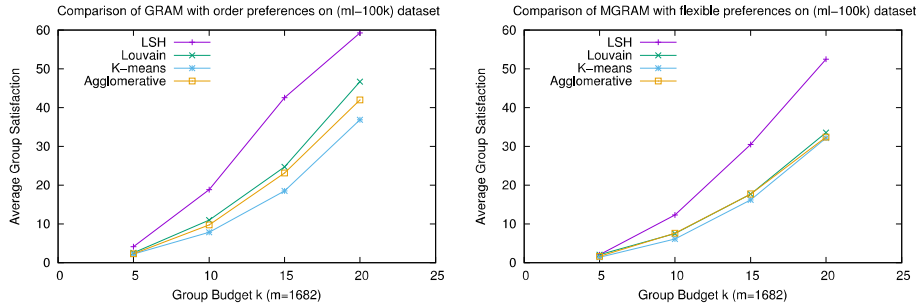
We apply different consensus functions in automatically detected groups and measure the performance by varying values of group budget (k), group size (n), and the number of items (m). The consensus functions generate group scores, which reflect the interests and preferences of all the group members. We also calculate the time complexity of the Min-Hash algorithm that maintains Jaccard Similarity profoundly in a reduced representation known as signature matrix.

Varying group budget (k):

Our experiments show that a higher value of k provides greater satisfaction. Figure 3.5 and Figure 3.6 depict the performance of LMM and MLMMP by setting the values of $n \leq 70$ and $m = 1682$, and this demonstrates that the average group satisfaction[5] score increases with k. Figure 3.3, Figure 3.5, Figure 3.8, Figure 3.9, Figure 3.11 and Figure 3.12 depict the similar characteristics.

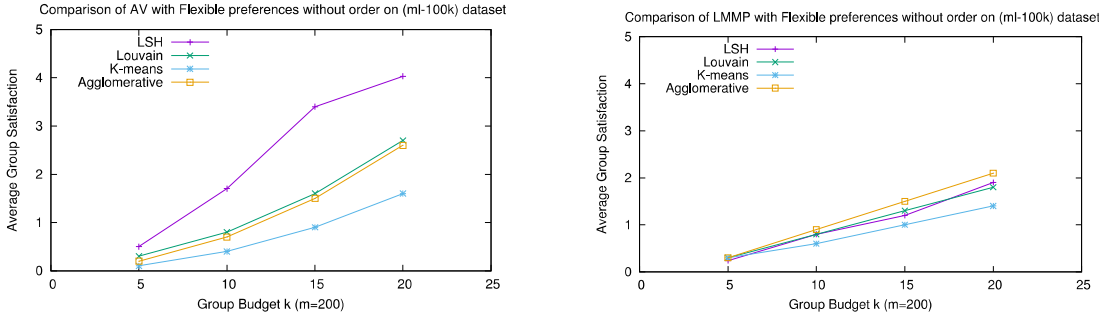
Figure 3.4 shows the result of AV and LMPP with flexible preferences without order on the ml-100k dataset. The term without order preferences signifies that generated recommendation vector does not consider the order of user preferences in the generated recommendation vector. The system applies the aggregation strategies to the user preferences of the group to produce the recommendation vector for group members. The recommendation steps involve sorting the items in increasing order at the end and inserting those items in a recommendation vector. The model calculates the average group satisfaction based on the produced recommendation vector. It is worth noting that generated recommendation in order user preferences holds a higher score than those without

[5]The average group satisfaction is the ratio of total group satisfaction to the number of users in a particular group.



(a) (b)

Figure 3.3: Average group satisfaction for a cluster ($30 \leq clusterSize \leq 40$) using a) GRAM and b) MGRAM on ml-100k dataset.



(a) (b)

Figure 3.4: Average group satisfaction of cluster ($30 \leq clusterSize \leq 40$) using a) AV and b) LMMP with flexible preferences without order on ml-100k dataset.

order user preferences (Table 3.7).

Varying the group size (n):

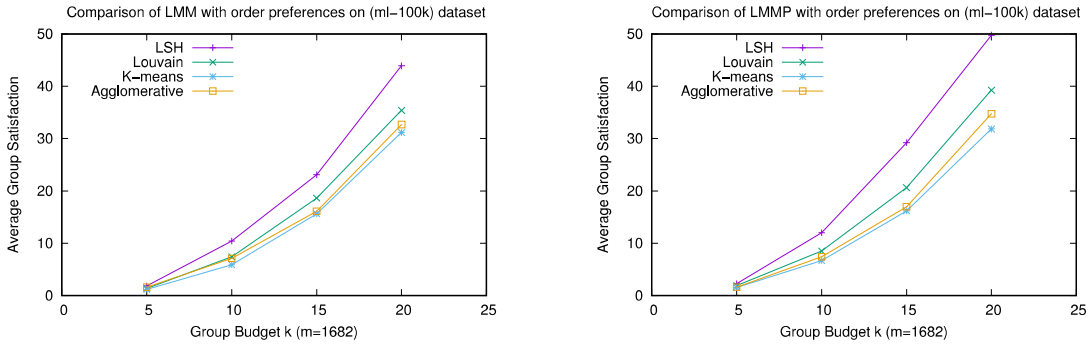
Increasing the number of users in a particular cluster and introducing flexibility in user preferences, we observed that the average group satisfaction $[1, 35]$ maximizes. Figure 3.7 shows the behaviour of LMM and LMMP algorithms by considering the order and unorder preferences for the Last.fm dataset. The outcome remains the same except when group budget (k)=5.

Varying the number of items (m):

The average group satisfaction increases with the increase in the number of items (m). LMM, LMMP, GRAM, MGRAM, HAM, and MHAM result in better group satisfaction

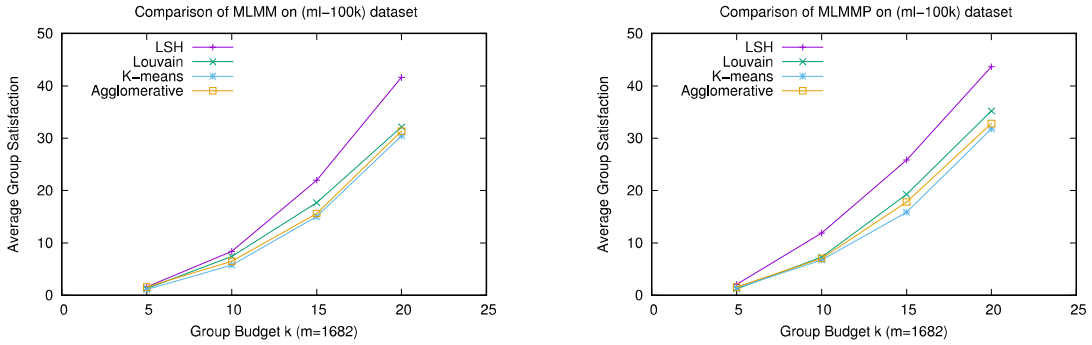
Table 3.7: Average group satisfaction of an automatically identified group of composed using Louvain algorithm of cluster size (n=60, m=17632) using LMM and LMMP on Last.fm dataset

Group Budget(k)	LMM		LMMP	
	With order	Without order	With order	Without order
5	4.6	4.6	5	4.6
10	16.2	15.2	16	15.2
15	29.3	27.5	27.9	27.5
20	49.1	46.04	46.04	46.04
25	69.1	63.2	69.1	63.2



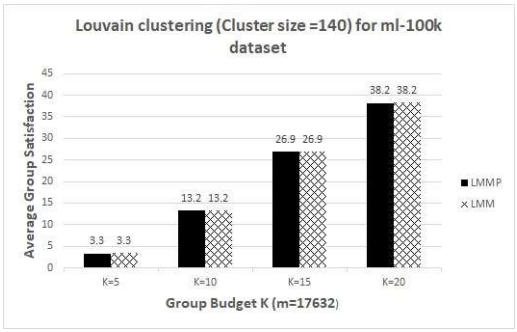
(a) (b)

Figure 3.5: Average group satisfaction of cluster size ($55 \leq clusterSize \leq 70$) using a) LMM and b) LMMP with order preferences on ml-100k dataset.

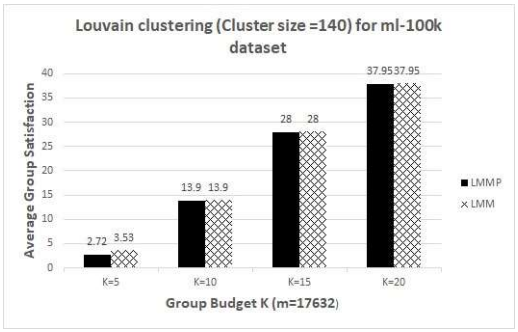


(a) (b)

Figure 3.6: Average group satisfaction of cluster size ($55 \leq clusterSize \leq 70$) using a) MLMM and b) MLMMP on ml-100k dataset.

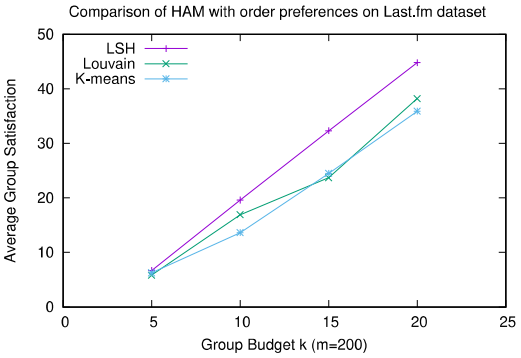


(a)

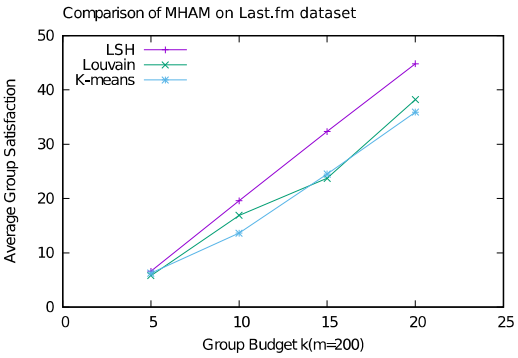


(b)

Figure 3.7: Average group satisfaction using LMM, LMMP a) with order and b) without order preferences on Last.fm dataset.

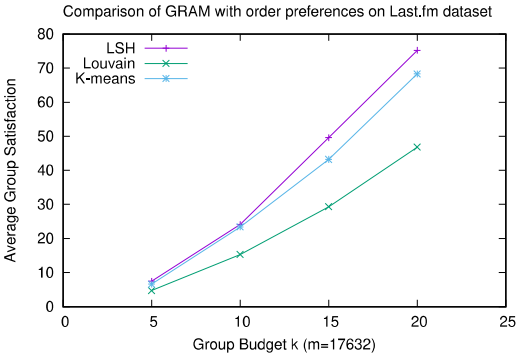


(a)

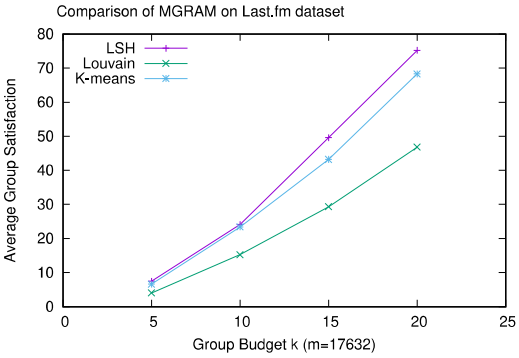


(b)

Figure 3.8: Average group satisfaction of cluster size ($50 \leq clusterSize \leq 60$) using a) HAM and b) MHAM on Last.fm dataset.

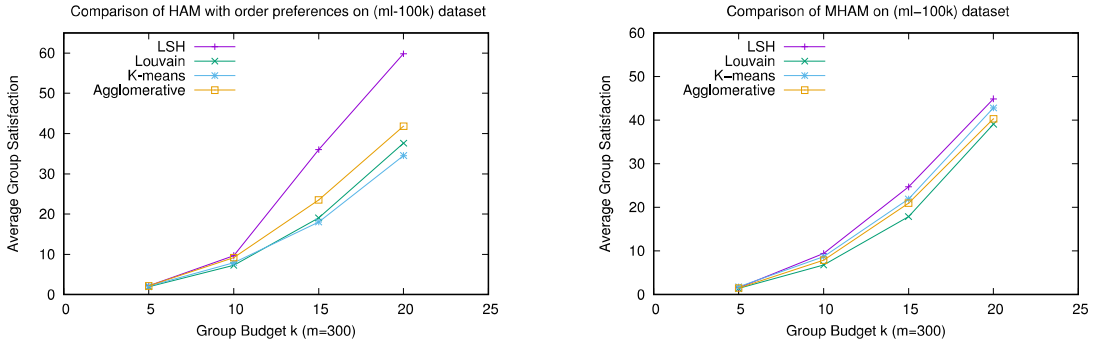


(a)



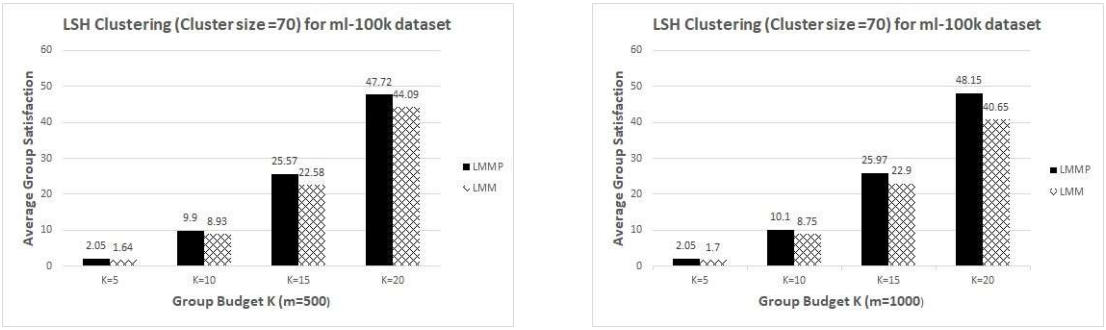
(b)

Figure 3.9: Average group satisfaction of cluster size ($50 \leq clusterSize \leq 60$) using a) GRAM and b) MGRAM on Last.fm dataset.



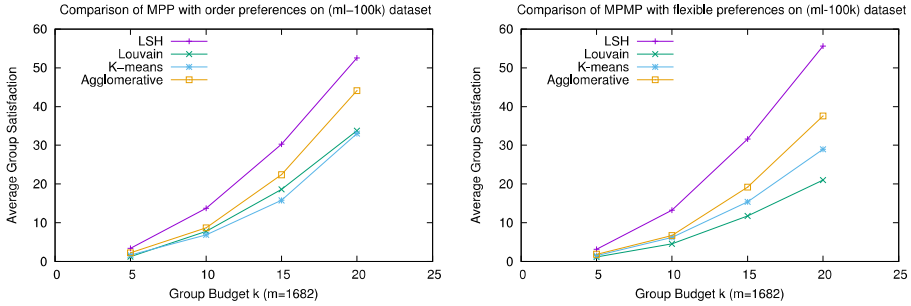
(a) (b)

Figure 3.10: Average group satisfaction of cluster size ($55 \leq clusterSize \leq 70$) using a) HAM and b) MHAM on ml-100k dataset.



(a) (b)

Figure 3.11: Average group satisfaction of cluster size=70 and a) m=500 and b) m=1000 using LMM and LMMP on ml-100k dataset.



(a) (b)

Figure 3.12: Average group satisfaction for a cluster ($30 \leq clusterSize \leq 40$) using a) MPP and b) MPMP on ml-100k dataset.

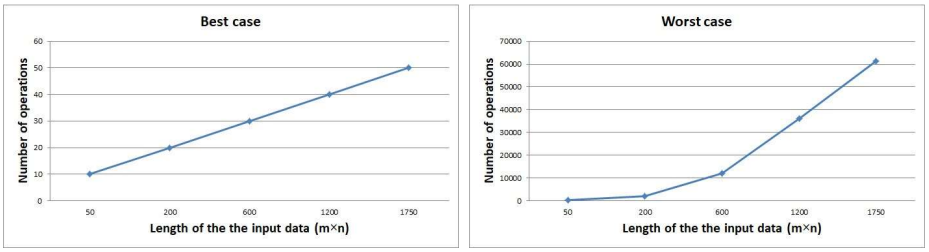


Figure 3.13: Runtime complexity analysis of MinHash technique.

by considering order and flexibility in user preferences. Figure 3.10 provides the result for HAM and MHAM by setting the parameters $n \leq 60$ and $m = 300$. We observe that this consensus function outperforms for smaller group sizes and items. Figure 3.11 shows that increasing m, it enhances the average group satisfaction.

Time complexity:

We calculate the time complexity of the MinHash algorithm. We check the efficacy of this algorithm by determining how fast this algorithm runs as the input size increases. In the MinHash technique, the input length is $m \times n$ where n represents the number of users, and m is the total number of items. It requires applying the l number of hash functions in this method. The best case time complexity to generate the signature matrix using this algorithm is $\Omega(n)$ (as l is constant). In the worst case scenarios, $l = m$. The worst-case time complexity to generate a signature matrix using the MinHash algorithm is $\mathcal{O}(m^2n)$. Figure 3.13 presents the runtime behaviour of the MinHash algorithm in the best case and worst case scenarios.

3.4 Summary

In this chapter, we employed a locality-sensitive hashing technique in group recommendations to quickly form the potential user groups while dealing with the curse of dimensionality. Existing clustering approaches do not consider dimensionality reduction while forming the groups. Though locality-sensitive hashing is not a traditional clustering approach, when composing an automatically identified group using this approach, we observe that LSH is very useful. LSH is efficient for automatic group detection in high-dimensional space. It eliminates the exhaustive search for similar users in the bucket by looking only

for the *candidate pairs* of users. Using most clustering approaches, the time complexity of finding similar users using a characteristic matrix is quadratic and now reduces to linear runtime. Nowadays, it is challenging to cope with the exponential growth of data, so scalability becomes an issue to recommend personalized items to a group of users. It could be handled by applying dimensionality reduction techniques more efficiently. The proposed system overcomes the scalability problem when dealing with massive user preference data.