Summarizing user-item matrix by group utility maximization

Wang, Yongjie; Wang, Ke; Long, Cheng; Miao, Chunyan

2023

Wang, Y., Wang, K., Long, C. & Miao, C. (2023). Summarizing user-item matrix by group utility maximization. ACM Transactions On Knowledge Discovery From Data, 17(6), 86-. https://dx.doi.org/10.1145/3578586

https://hdl.handle.net/10356/166437

https://doi.org/10.1145/3578586

Summarizing User-Item Matrix By Group Utility Maximization

YONGJIE WANG, Nanyang Technological University, Singapore
KE WANG, Simon Fraser University, Canada
CHENG LONG, Nanyang Technological University, Singapore
CHUNYAN MIAO, Nanyang Technological University, Singapore

A user-item utility matrix represents the utility (or preference) associated with each (user, item) pair, such as citation counts, rating/vote on items or locations, and clicks on items. A high utility value indicates a strong association of the pair. In this work, we consider the problem of summarizing strong association for a large user-item matrix using a small summary size. Traditional techniques fail to distinguish user groups associated with different items (such as top-l item selection) or fail to focus on high utility (such as similarity based subspace clustering and biclustering). We formulate a new problem, called Group Utility Maximization, to summarize the entire user population through k user groups and l items for each group; the goal is to maximize the total utility of selected items over all groups collectively. We show this problem is NP-hard even for $l = 1$. We present two algorithms. One greedily finds the next group, called Greedy algorithm, and the other iteratively refines existing k groups, called k-max algorithm. Greedy algorithm provides the $(1 - \frac{1}{e})$ approximation guarantee for a nonnegative utility matrix, whereas k-max algorithm is more efficient for large datasets. We evaluate these algorithms on real-life datasets.

CCS Concepts: • **Information systems** → **Data mining**; **Summarization**; • **Theory of computation** → **Design and analysis of algorithms**.

Additional Key Words and Phrases: Data summarization, Greedy algorithm, k-max algorithm, Group utility maximization

1 INTRODUCTION

In the zettabyte era, massive volumes of data are produced every day, and it is problematic to analyze, retrieve, and comprehend data at such scales. To mitigate this problem, data summarization techniques [1, 20] provide a concise, compact, yet informative representation of original data. These techniques include top-l selection [21], histograms [20], clustering [31], sampling [13, 32], matrix decomposition [2], frequent itemset mining [7], principal component analysis (PCA) [28] and wavelets [34].

1.1 Motivation

In this paper, we consider summarizing a utility matrix with rows representing users and columns representing items. Each entry in the user-item utility matrix represents the utility (or preference) associated with a (user,item) pair. The terms of "user" and "item" are only for convenience and can represent any two classes of objects that interact in some way, such as author-to-paper citation, customer-to-item purchases, user's rating/vote on items or locations, user's website clicks, weighted edges in a social network, gene expression scores under conditions and samples' feature importance for model's prediction. In many real life applications, a high utility is more interesting because it indicates a strong association, e.g., a high citation count, a high rating, a presence of purchase, a high level of gene expression, importance of a feature, etc.

Authors' addresses: Yongjie Wang, yongjie002@e.ntu.edu.sg, Nanyang Technological University, N4-B3-06, 50 Nanyang Avenue, Singapore, 639798; Ke Wang, wangk@cs.sfu.ca, Simon Fraser University, 8888 University Drive, Burnaby, British Columbia, Canada, V5A 1S6; Cheng Long, c.long@ntu.edu.sg, Nanyang Technological University, N4-2c-117a, 50 Nanyang Avenue, Singapore, 639798; Chunyan Miao, ascymiao@ntu.edu.sg, Nanyang Technological University, N4-B3-06, 50 Nanyang Avenue, Singapore, 639798.

Table 1. The user-item matrix $\mathcal{D}$ contains the feature importance (i.e., IG) values for the survival probability $F(x)$ of six Titanic correctly predicted survivors x, where F is a black-box ML model. The IG value of the feature x_i for a survivor x represents the contribution of x_i to the prediction $F(x)$ and x_i with a larger IG value is more influential for x's prediction. With $k = 2$ and $l = 2$, GUM partitions the six survivors into two groups, i.e., $\{U_1, U_2, U_3\}$ and $\{U_4, U_5, U_6\}$, and selects two most influential features for each group, marked by the green and pink colors.

	Pclass	Sex	Age	SibSp	Parch	Fare	C	Q	S
U_1	1.30	1.49	-0.20	-0.12	-0.05	0.28	-0.24	0.01	0.14
U_2	1.12	1.40	0.06	0.15	-0.04	0.31	0	-0.02	-0.10
U_3	1.22	1.49	0.15	0.25	-0.14	0.27	-0.01	-0.02	-0.12
U_4	-0.35	1.01	0.82	0.16	0.29	0	-0.02	0.01	-0.19
U_5	-0.34	0.88	0.92	0.09	0.41	-0.04	-0.16	0.01	0.35
U_6	0.49	1.34	0.72	-0.14	0.11	0.03	-0.03	-0.01	-0.16

The main question studied in this work is: given a user-item utility matrix, how to summarize the high utility interactions of the whole user pool using a small summary. Take the MovieLens data [4] as an example where millions of users give ratings to hundreds of thousands of movies. Scanning through individual user's ratings does not give a high level overview on what movies users like due to the large number of users and movies. Traditional summarization techniques fail to address our summary that focuses on high utility. For example, the top-l item selection [21] summarizes all users using the same l items though typically different user groups may prefer different movies, therefore, the top-l items fail to maximize the utility for different user groups; biclustering [24] groups users and items according to the similarity of cells in a matrix, which fails to focus on high utility.

With the above in mind, we define a new summarization problem, called *Group Utility Maximization (GUM)*: given a user-item matrix and integers k and l, we want to partition users into k groups and select l items for each group so that the sum of utility of selected items over all groups is maximized. In other words, we want to summarize all users through k groups with each group having its own top l items. The selected items for each group summarize the preferences of the users in the group. Since the data analyst has to read l items for each group, $k \times l$ represents the summary size and k, l are usually small integers. Note that users are strictly partitioned but items selected for different groups can be overlapped. The objective is to group the users such that the sum of the utilities of selected items over all groups is maximized. This objective is different from clustering that is similarity based, and different from max-sum submatrix that extracts certain submatrices that do not always contain all users. See more discussion on related works in Section 2.

Consider the MovieLens example again. Despite millions of users and hundreds of thousands of movies and billions of ratings, GUM will find a summary of size $k \times l$, which summarizes all users in k groups with each group being summarized by l top rated items. The case of $k = 1$ summarizes all users in one group by the top-l items. In general, there may be $k > 1$ groups of users and each group has its distinct preferred items. One application of GUM is summarizing the explanation of a black-box model's prediction, as shown in the following example.

Example 1. Consider a neural network $F(x)$ for predicting the surviving probability of Titanic survivors [1], where each survivor x has the features x_i: Pclass (cabin class), Sex, Age, Sibsp (number of siblings/spouses aboard), Parch (number of parents/children aboard), Fare, and Port of embarkation denoted by C, Q, S. [35] proposes the Integrated Gradient (IG) to measure the attribution (i.e.,

[1] https://www.kaggle.com/c/titanic

importance) of each feature x_i to the prediction $F(x)$. Table 1 shows the IG values of six survivors as a user-item matrix $\mathcal{D}$. A high IG value means that the feature in the corresponding column is more important for $F(x)$ for the survivor x in the corresponding row. Suppose that the data analyst wants to get a quick grasp of important features for $F(x)$ for all survivors x, but this is not easy because important features generally vary for different survivors, especially when there is a large number of users and features.

Through the GUM problem, the data analyst can get a high level summary of important features, as shown by the two colors in Table 1 where the survivors are partitioned into $k = 2$ groups and each group is summarized by $l = 2$ most influential features. The choices of these groups and selected items will maximize the sum of the IG values over the 12 colored entries. The group in green has Pclass and Sex as the two most influential factors for the surviving probability, and the group in pink has Sex and Age as two most influential factors for the surviving probability. Importantly, the data analyst only needs to read 6×2 IG values to get a high level overview about important features, regardless of the number of survivors and features in the matrix. Such a group level summary provides a concise and easy-to-understand explanation of important features for all survivors. □

1.2 Contributions

The major contribution of this paper is summarized as follows:

(1) (Section 3) We define the GUM problem for a user-item utility matrix $\mathcal{D}$ and show the NP-hardness of the problem and the monotonicity and submodularity of the objective function behind the problem.
(2) (Section 4) We present an algorithm called Greedy, which finds one group that maximizes the marginal gain at a time. This algorithm provides a $(1 - \frac{1}{e})$ approximation factor when the matrix is non-negative.
(3) (Section 5) We present a heuristic k-max algorithm that iteratively refines the k groups. In each iteration, it performs *Assignment* and *Update* steps to maximize the utility from k initial groups. Compared with the Greedy algorithm, the k-max algorithm is more scalable for larger datasets since it does not evaluate an exponential number of l-itemsets as the Greedy algorithm does.
(4) (Section 6) We evaluate the usefulness and efficiency of the proposed summarization methods on real life datasets. The study shows that 1) the two proposed methods produce better utility than existing summarization techniques, 2) our summary provides an easy-to-understand overview of the whole dataset, and 3) the k-max algorithm achieves empirically comparable utility but is more efficient than the Greedy algorithm for large datasets.

Among these contributions, Item 1 (partial), Item 3, and Item 4 (partial) have been covered in our prior work [37], and Item 1 (partial), Item 2, and Item 4 (partial) are newly covered in this extended paper. Specifically, for Item 1, the NP-hardness, monotonicity and submodularity results are newly introdued in this paper, for Item 2, it is completely new in this paper, and for Item 4, we conducted more experiments, including the comparison of the new Greedy algorithm, more utility evaluations in Section 6.3, and scalability evaluations on the new dataset MovieLens in Section 6.4.

2 RELATED WORK

In this section, we discuss the differences of our work from several related works, namely, clustering, subgroup discovery, max-sum submatrix, preference learning and data summarization.

Clustering [3, 30, 36] is the task of grouping a set of objects such that objects in the same group are more similar to each other than to those in different groups. *Subspace clustering* [29]

groups objects in subspace. Motivated by gene expression analysis for finding submatrices of genes exhibiting similar behaviors, *biclustering* [11], a.k.a. *co-clustering* [17], finds genes subsets (rows) with similar expression values (columns). See the survey papers [24, 33] for more details of clustering methods. All these problems rely on a similarity measure for a pair of objects. Our GUM does not use any similarity measure because the goal is to maximize the utility sum of groups, not similarity of objects. In Example 1, features C and Q (ports of embarkation) are more similar than Pclass and Sex for the first group but the latter is more interesting for explaining the prediction because higher IG values mean more importance to the survival prediction. Due to this difference in the objective, existing clustering works cannot address our problem.

Subgroup discovery [19, 38] is concerned with finding descriptive associations (usually represented as rules) among attributes with respect to a target property of interest. For example, if the success rate of an operation is 30% over all patients and if the success rate for female patients under 30 and with drug A intake is 90%, these algorithms will identify the rule "Gender = Female AND Age < 30 AND Drug = A -> Operation = SUCCESS", which describes an unusually high operation success rate for a subgroup. *Exceptional model mining* [22] allows more than one target and looks for unusual target interaction. *Exceptional preferences mining* [15] searches for subgroups with deviating preferences. For example, if a subgroup ranks tekka-maki consistently in the top 3 while the majority in the dataset ranks it in the last 3, this measure will find the subgroup to be very interesting. All these works find interesting *subsets* of data with respect to some pre-selected target properties (e.g., operation success rate and tekka-maki). In contrast, GUM summarizes the *whole* dataset by partitioning the dataset into groups and maximizing the utility sum for groups.

Max-sum submatrix [9, 10] targets highly expressed subsets of genes and of samples by identifying a submatrix with the maximum sum. [17] finds k submatrices by constraint programming and [8] adds the submatrix disjointness constraint. While similar in maximizing the sum of entries, there are key differences between these works and ours. Motivated by the objective of summarization, our groups cover all users and are nonoverlapping in rows but not in columns. In contrast, the max-sum submatrix solution allows overlapping of both rows and columns, and does not require to cover all rows. The max-sum submatrix problem assumes that the matrix contains both positive and negative entries. For a nonnegative matrix, which is common for ratings, votes, and implicit feedback, their problem will return the entire matrix as the solution due to the maximum sum, which is useless for our summarization. The authors of [9, 10] suggested to subtract all entries by a constant to make the matrix contain both positive and negative values, but did not give a clue on what constant should be used.

Preference learning [16] aims to build a global predictive model to predict the order of preferences for new cases, and *recommender system* [14] seeks to predict the rating that a user would give to an unseen item. Our GUM is designed for a better understanding of preferences of existing users, instead of prediction for future users.

Data summarization [6, 27, 39] targets to produce a compact summary of original data by choosing the most informative and representative content, e.g., keyframes [27], keywords [39] and image prototypes [6]. Due to different problem contexts and objectives, these summarization algorithms cannot address our GUM problem that maximizes the utility of selected entries. To the best of our knowledge, our work is the first to summarize a dataset by k groups with each group being summarized by top-l interesting items.

3 PROBLEM STATEMENT

We now formally define the GUM problem and show the NP-hardness, the monotonicity, and the submodularity of this problem. Some frequently used notations are given in Table 2.

Table 2. Frequently used notations.

Symbol	Description
$\mathcal{D} \in \mathbb{R}^{N \times M}$	dataset of N rows over M columns $1, ..., M$
r_{ic}	the value of item c of user r_i in $\mathcal{D}$
(k, l)	problem parameters: group number and itemset size
D_j	a subset of rows
$\{D_1, ..., D_k\}$	a partitioning of $\mathcal{D}$
X_j	l-itemset, i.e., a set of l items
$\{X_1, ..., X_k\}$	a collection of l-itemsets
$h(\cdot)$	the utility function over a collection of l-itemsets on a row
$f(\cdot)$	the utility function over a collection of l-itemsets on $\mathcal{D}$

3.1 Definitions

In this section, we consider a user-item matrix $\mathcal{D} \in \mathbb{R}^{N \times M}$, consisting of N rows for users $\{r_1, ..., r_N\}$ and M columns for items $\{1, ..., M\}$. r_{ic} denotes user r_i's score on item c (e.g., vote, rating, etc.) and a larger score represents a higher utility or preference. The only assumption is that the addition operation is meaningful for entry values over rows and columns. For example, r_{ic} representing ratings in 5-star scales is additive (because the sum returns the total number of stars), and r_{ic} in the log scale is not additive. For a set X of l items, a.k.a. l-itemset, and a user r_i, we define $g(r_i, X) \equiv \sum_{c \in X} r_{ic}$ as the total score of r_i over the items in X. For a collection of l-itemsets $\mathbb{X} = \{X_1, ..., X_k\}$, named *summary*, where $X_i \neq X_j$ for $i \neq j$, we define $h(r_i, \mathbb{X}) \equiv \max_{X_j \in \mathbb{X}} g(r_i, X_j)$ and define the *utility* of $\mathbb{X}$ as

$$f(\mathbb{X}) \equiv \sum_{i=1}^{N} h(r_i, \mathbb{X}) \tag{1}$$

In other words, $f(\mathbb{X})$ is the total utility over all users by assigning each user r_i to the X_j that maximizes $g(r_i, X_j)$. Therefore, $\mathbb{X}$ induces a partitioning on the users. Let $f(\emptyset) \equiv 0$. Note that $h(r_i, \mathbb{X})$ can also be defined using min instead of max operator. In the rest of this work, we consider only maximization because minimization can be performed by negating all utility values first.

Definition 1 (GUM Problem)**.** Given a user-item matrix $\mathcal{D}$, and numbers k and l, find a collection of distinct l-itemsets $\mathbb{X}^* = \{X_1, ..., X_k\}$ such that

$$\mathbb{X}^* = \arg\max_{\mathbb{X}} f(\mathbb{X}) \tag{2}$$

We write $f(\mathbb{X}^*)$ as $f^*(k, l)$ or simply f^* if k and l are understood. □

We say that a partitioning $D_1, ..., D_k$ of $\mathcal{D}$ is *induced* by $\mathbb{X}$ where D_j contains a user r_i if $h(r_i, \mathbb{X}) = g(r_i, X_j)$, that is, D_j contains all users r_i such that X_j gives the maximum $g(r_i, X_j)$, i.e., the maximum sum of r_i's scores on the items in X_j. This gives rise to the following equivalent definition of GUM.

Definition 2 (GUM Problem (Alternative))**.** Given a user-item matrix $\mathcal{D}$ and numbers k and l, find distinct l-itemsets $\mathbb{X} = \{X_1, ..., X_k\}$ such that for the partitioning $D_1, ..., D_k$ induced by $\mathbb{X}$,

$$\sum_{j=1}^{k} \sum_{r_i \in D_j} g(r_i, X_j)$$

is maximized. □

Intuitively, $(D_1, X_1), ..., (D_k, X_k)$ is a group level summary where each X_j represents the l most preferred items (in terms of largest sum) over the users in the group D_j. The group number k and description size l represent the summary size, serving as the budget on the cost for reading the summaries. For example, if $\mathcal{D}$ stores ratings/votes, the GUM solution summarizes what items the users would like through a summary of size $|X_1| + \cdots + |X_k| = k \times l$. Note that this size is independent of the data size $|\mathcal{D}|$. Similar to the cluster center that summarizes the location of points in each cluster, the l-itemset X_j summarizes the preferred items for each group D_j. The difference is that, instead of being the mean of points, X_j is the top-l items (in terms of score sum) in the group. It is possible that some D_j is empty, which means that fewer than k groups are sufficient to achieve the same utility as k groups.

Typically, l is a small number, say 1 to 5, as too many items would overwhelm the human analyst. k is in the range $[1, N]$. $k = N$ summarizes each user by personalized l items, whereas $k = 1$ summarizes all users by the same set of l items and corresponds to the standard top-l items selection over the whole data set. A smaller k produces larger groups , thus, a larger variance of scores in such groups. A larger k leads to more customized preferences for smaller groups but the summary size increases. k can be specified by the analyst or determined similar to determining the cluster number k for k-means clustering methods (i.e., gradually increasing k until the increase of f^* slows down significantly). Finally, note that for the same l, f^* never decreases as k increases, and for the same k, the average utility of selected items, i.e., $\frac{f^*}{N \times l}$, never increases as l increases (because lower utility items are selected as l increases).

The maximization objective in Definition 1 is sufficient for modeling other requirements in practice. For example, for the 5-star rating scale, if both a low rating close 1 and a high rating close to 5 are interesting, we need to summarize users in terms of both high ratings and low ratings. In this case, we can shift the 5-star scale to the symmetric scale $[-2, 1, 0, 1, 2]$ by subtracting the medium rating 3 from every known rating and consider $\mathcal{D}$ defined by the absolute values of shifted ratings in Definition 1. Then f represents the sum over the distances from the medium rating 0 and the GUM problem will summarize the users into groups by the items that have most extreme ratings.

3.2 Properties

With $\binom{M}{l}$ possible l-itemsets, there are $\binom{\binom{M}{l}}{k}$ possible summaries $\{X_1, ..., X_k\}$. This number grows quickly as M, l, k grow. The exception cases are $k = 1$, which coincides with the top-l items selection on the whole dataset, and $k = N$, which coincides with the top-l items selection for each user. In general, even for the strict case $l = 1$, GUM problem is NP-hard, as shown below.

Theorem 1. *The GUM problem is NP-hard for $l = 1$.*

Proof. Consider the following NP-hard *maximum coverage problem* [12]: Given a number k and a collection of sets $S = \{S_1, \cdots, S_m\}$, find a subset $S' \subseteq S$ of sets, such that $|S'| \leq k$ and $|\cup_{S_j \in S'} S_j|$ is maximized.

We can construct an instance of GUM problem from the maximum coverage problem in polynomial time such that $\mathbb{X}$ is a solution to the GUM problem if and only if S' is a solution to the maximum coverage problem, where S' is a subset of S constructed from $\mathbb{X}$. Therefore, if there is a polynomial time algorithm for the GUM problem, we also have a polynomial time algorithm for the maximum coverage problem.

We define the instance $< N, M, \mathcal{D}, k, l >$ for the GUM problem as follows. Let $N = |\cup_{S_j \in S} S_j|$, $M = |S|$, k be same as in the maximum coverage problem, and $l = 1$. $\mathcal{D}$ contains one row for each element in $\cup_{S_j \in S} S_j$ and one item (column) for each S_j in S, such that the entry for row r

and item c contains 1 if the corresponding element for the row is in the S_j corresponding to the item c, otherwise 0. Note that with $l = 1$, every l-itemset is a single item and there is an 1-to-1 correspondence between a S_j in S and a l-itemset X_j.

Consider any collection of l-itemsets $\mathbb{X} = \{X_1, ..., X_k\}$ for the above GUM problem. For each row r_i in $\mathcal{D}$, $h(r_i, \mathbb{X}) = \max_{X_j \in \mathbb{X}} g(r_i, X_j)$. With $l = 1$, each X_j is a single item and $h(r_i, \mathbb{X}) = 1$ if r_i has 1 for some X_j in $\mathbb{X}$, otherwise $h(r_i, \mathbb{X}) = 0$. Therefore, $f(\mathbb{X})$ (Equation (1)) is equal to the number of rows that have 1 for some X_j in $\mathbb{X}$. Define S' to be the set of S_j's corresponding to the X_j's in $\mathbb{X}$, $f(\mathbb{X}) = |\cup_{S_j \in S'} S_j|$. Therefore, $f(\mathbb{X})$ is maximized if and only if $|\cup_{S_j \in S'} S_j|$ is maximized, in other words, $\mathbb{X}$ is a solution to the GUM problem if and only if S' is a solution to the maximum coverage problem. The only difference is that S' may contain redundant S_j's whose removal from S' does not affect $\cup_{S_j \in S'} S_j$; such S_j's correspond to the X_j's in $\mathbb{X}$ that have empty groups. □

Next, we show two properties of f that are useful for proving an approximation bound of our solution.

Theorem 2 (Monotonicity of f). *For a nonnegative $\mathcal{D}$ (i.e., $\mathcal{D} \in \mathbb{R}_{\geq 0}^{N \times M}$),*

- *(i) if $\mathbb{X} = \{X_1, ..., X_k\}$ and $\mathbb{X}' = \mathbb{X} \cup \{X_{k+1}\}$, then $f(\mathbb{X}') \geq f(\mathbb{X})$, and*
- *(ii) $f^*(k+1, l) \geq f^*(k, l)$.*

Proof. (i). Recall $f(\emptyset) = 0$. The nonnegativity of $\mathcal{D}$ implies $g(r_i, X_1) \geq 0$ for any l-itemset X_1, so $f(\{X_1\}) \geq f(\emptyset)$. Then (i) follows because having an additional X_{k+1} gives one more choice of X_j for the max function of computing $h(r_i, \mathbb{X})$, thus, never decreases the value of f. (ii) If $\mathbb{X} = \{X_1, ..., X_k\}$ is the solution of $f^*(k, l)$ and if $\mathbb{X}' = \mathbb{X} \cup \{X_{k+1}\}$, then $f^*(k+1, l) \geq f(\mathbb{X}') \geq f^*(k, l)$. The second inequality follows from (i). □

(ii) implies that $f^*(k, l)$ is not worse than $f^*(1, l)$, which is the utility of the standard top-l selection over the whole dataset.

We say that a function $f : 2^{\mathcal{N}} \rightarrow \mathbb{R}$ is *submodular* if $f(A \cup \{u\}) - f(A) \geq f(B \cup \{u\}) - f(B)$, $\forall A \subseteq B \subseteq \mathcal{N}$, $u \in \mathcal{N} \backslash B$, $\mathcal{N}$ is a ground set of elements. In other words, f is submodular if the marginal gain of adding an element to a subset A is no less than the marginal gain of adding this element to any superset of A. In our context, the universe $\mathcal{N}$ is the set of all l-itemsets. Next, we show that f is submodular for a general $\mathcal{D}$.

Theorem 3 (Submodularity of f). *For a general $\mathcal{D}$, the function f (defined by Equation (1)) is submodular.*

Proof. Let $\mathbb{X}$ be a set of l-itemsets and $\mathbb{X}'$ be a subset of $\mathbb{X}$, i.e., $\mathbb{X}' \subset \mathbb{X}$. In addition, let X be an l-itemset that is not in $\mathbb{X}$. Here, we consider each user separately. For a arbitrary user r_i, we show that the gain of adding X to $\mathbb{X}'$ is at least that of adding X to $\mathbb{X}$ (both wrt r_i), i.e.,

$$h(r_i, \mathbb{X}' \cup \{X\}) - h(r_i, \mathbb{X}') \geq h(r_i, \mathbb{X} \cup \{X\}) - h(r_i, \mathbb{X}) \tag{3}$$

Then summing up on both sides over all users r_i, we obtain

$$f(\mathbb{X}' \cup \{X\}) - f(\mathbb{X}') \geq f(\mathbb{X} \cup \{X\}) - f(\mathbb{X}) \tag{4}$$

which implies that function f is submodular. Therefore, what remains is to verify Equation (3).

Let X_i be the l-itemset in $\mathbb{X}$, for which the utility of r_i is the largest, i.e., $X_i = \arg\max_{X' \in \mathbb{X}} g(r_i, X')$. We consider two cases depending on whether X_i is in $\mathbb{X}'$ or not.

Case 1 ($X_i \in \mathbb{X}'$): In this case, only $\mathbb{X}'$ need to be considered when computing the gain of adding X to $\mathbb{X}$, which is exactly the case when computing the gain of adding X to $\mathbb{X}'$. Therefore, we have

$$h(r_i, \mathbb{X} \cup \{X\}) - h(r_i, \mathbb{X}) = h(r_i, \mathbb{X}' \cup \{X\}) - h(r_i, \mathbb{X}') \tag{5}$$

which implies that Equation (3) holds in Case 1.

Case 2 ($X_i \in \mathbb{X} \backslash \mathbb{X}'$): This case implies

$$h(r_i, \mathbb{X}') \leq g(r_i, X_i) \tag{6}$$

We consider two sub-cases based on r_i's utility on X.

Case 2.1 ($g(r_i, X) > g(r_i, X_i)$): In this sub-case, we have

$$h(r_i, \mathbb{X} \cup \{X\}) = h(r_i, \mathbb{X}' \cup \{X\}) = g(r_i, X) \tag{7}$$

$$h(r_i, \mathbb{X}) = g(r_i, X_i) \geq h(r_i, \mathbb{X}') \tag{8}$$

Equation (7) and (8) imply that Equation (3) holds in Case 2.1.

Case 2.2 ($g(r_i, X) \leq g(r_i, X_i)$): In this sub-case, we have

$$h(r_i, \mathbb{X} \cup \{X\}) - h(r_i, \mathbb{X}) = g(r_i, X_i) - g(r_i, X_i) = 0 \tag{9}$$

$$h(r_i, \mathbb{X}' \cup \{X\}) - h(r_i, \mathbb{X}') \geq 0 \ (h \text{ is monotone}) \tag{10}$$

Equation (9) and (10) imply that Equation (3) holds in Case 2.2.

In conclusion, Equation (3) holds for any user r_i on both cases. Summing up over all users, we can deduce that the function f is also submodular. □

We can also explain the intuition of the submodularity using $f(\{X_1\} \cup \{X\}) - f(X_1) \geq f(\{X_1, X_2\} \cup \{X\}) - f(\{X_1, X_2\})$, where X is a l-itemset different from X_1 and X_2. Recall that $f(\mathbb{X})$ assigns each row r_i to the X_j in $\mathbb{X}$ that maximizes $g(r_i, X_j)$. The left-hand-side of the inequality is the marginal gain of reassigning some rows r_i from X_1 to X, and the right-hand-side is the marginal gain of reassigning some rows r_i from $\{X_1, X_2\}$ to X. Let g_1 and g_2 be the gains in these two cases, respectively. There must be a nonnegative gain, denoted g', for moving r_i from X_1 to $\{X_1, X_2\}$, and $g_1 = g' + g_2$. Therefore, $g_1 \geq g_2$.

In the next two sections, we present two algorithms for solving the GUM problem for a matrix $\mathcal{D}$.

4 GREEDY ALGORITHM

In this section, we first propose the Greedy algorithm that greedily finds a set of l-itemsets successively. This algorithm achieves the $(1 - \frac{1}{e})$ approximation factor of optimal solutions for a non-negative matrix as shown in Section 4.1. Next, we introduce two strategies to speed up the Greedy algorithm in Section 4.2.

With the input parameters $k, l, N, M, \mathcal{D}$, Greedy algorithm in Algorithm 1 starts with an empty set $\mathbb{X}$, adds one l-itemset X_j at a time that maximizes the marginal gain of f, $\Delta f(X_j|\mathbb{X}) \equiv f(\mathbb{X} \bigcup \{X_j\}) - f(\mathbb{X})$, until k l-itemsets are added. After finding $\mathbb{X}$, the corresponding $D_1, ..., D_k$ can be induced by $\mathbb{X}$ in one additional pass over the data as discussed in Section 3. In each iteration, the $\arg\max_{X_j \in \mathcal{X} \backslash \mathbb{X}}$ operation requires evaluating the marginal gains for $\binom{M}{l}$ l-itemsets, and each l-itemset takes $O(N \cdot l)$ time to check the utility for N users. For k iterations, the overall time complexity is $O(k \cdot \binom{M}{l} \cdot N \cdot l)$. Note that computing $\arg\max_{X_j \in \mathcal{X} \backslash \mathbb{X}}$ does not require materializing $\mathcal{X}$. Instead, we can enumerate the l-itemsets using l nested loops where each loop iterates over all items $\{1, \cdots, M\}$. This method enumerates only one l-itemset at a time, requiring the constant space $O(1)$, plus the space for the k l-itemsets of $\mathbb{X}$ and the space for the input matrix. Therefore, the overall space complexity is $O(N \cdot M)$.

Algorithm 1: Greedy algorithm

Input: A matrix $\mathcal{D}$, group number k, itemset size l
Output: $\mathbb{X}$
Notation: Let $\mathcal{X}$ as the collection of all l-itemsets;
initialize $\mathbb{X}$ as an empty set;
for $i = 1;\ i \leq k;\ i = i + 1$ **do**
  $X \leftarrow \arg\max_{X_j \in \mathcal{X} \setminus \mathbb{X}} \Delta f(X_j|\mathbb{X})$;
  $\mathbb{X} \leftarrow \mathbb{X} \bigcup \{X\}$;
return $\mathbb{X}$

4.1 Approximation Factor

For a nonnegative $\mathcal{D}$, the approximate solution by Greedy algorithm has its quality guaranteed to be not far away from that of the optimal one. We present this result in the following theorem.

Theorem 4. *For a nonnegative $\mathcal{D}$, Greedy algorithm (Algorithm 1) provides a $(1 - \frac{1}{e})$-factor approximation for the GUM problem, where e is the natural logarithmic base.*

Proof. The result holds since the f function is monotone and submodular (Theorem 2 and Theorem 3) and according to [26], a simple Greedy algorithm would provide a $(1 - \frac{1}{e})$-factor approximation for maximizing a monotone and submodular function f with $f(\emptyset) = 0$ subject to a cardinality constraint, i.e., $\max_{\mathbb{X}} f(\mathbb{X})$ s.t. $|\mathbb{X}| \leq k$ in our context. □

Theorem 4 shows $(1 - \frac{1}{e})$-factor approximation for a nonnegative matrix $\mathcal{D}$. In practice, a nonnegative $\mathcal{D}$ is common, such as rating/vote, citation count, presence of purchase, etc. In the case of a general $\mathcal{D}$ containing both positive and negative values, Greedy algorithm still works but we cannot claim the $(1 - \frac{1}{e})$ approximation factor. While adding a positive constant to every entry can make the matrix nonnegative, the $(1 - \frac{1}{e})$ approximation will be for the the transformed nonnegative matrix, not the original matrix. Replacing all negative scores with zeros also makes the matrix nonnegative, but this has the effect of ignoring the penalty of negative scores, which changes the problem statement.

Normalization indeed can be done to reduce the storage and avoid data overflow due to sum of large values. In theory, however, no normalization is required, especially in the case that no approximation factor is required. The Greedy algorithm can be run on the normalized matrix (to obtain a nonnegative matrix), say by shifting then scaling, in which case the $(1 - \frac{1}{e})$ approximation factor will be for the normalized matrix instead of the original matrix. In general, the Greedy algorithm is not affected by normalization in terms of scaling only (e.g., divide all values by the maximum value).

4.2 Implementation

The costly step of Greedy algorithm is finding the l-itemset that has the greatest marginal gain from all those l-itemsets that have not been selected at each iteration. For better efficiency, we adopt two existing strategies, namely *Lazy-forward* [23] and *Random-sampling* [25], and discuss how to integrate them together efficiently.

Lazy-forward. The idea is to leverage the submodularity of f that the marginal gain Δf of X_j never increases by growing $\mathbb{X}$. Based on this property, we can prune the evaluations of Δf for a X_j if its upper bound on Δf is smaller than the greatest Δf evaluated so far. In particular, we maintain an upper bound of the marginal gain for each l-itemset X_j and check the l-itemsets in the decreasing order of their upper bounds using a priority-queue, if its upper bound is larger than

Algorithm 2: Stochastic-Greedy algorithm

Input: A matrix $\mathcal{D}$, group number k, itemset size l, sampling factor ϵ.
Output: $\mathbb{X}$
Notation: Let $\mathcal{X}$ as a collection of all l-itemsets;
initialize $\mathbb{X}$ as an empty set;
for $i = 1;\ i \leq k;\ i = i + 1$ **do**
  /* Randomly sampling from $\mathcal{X} \backslash \mathbb{X}$ */
  $x \leftarrow min(\frac{|\mathcal{X}|}{k} log(\frac{1}{\epsilon}), |\mathcal{X}|)$;
  $R \leftarrow$ randomly sample x distinct numbers from $\{1, \cdots, |\mathcal{X}|\} - \{j \mid X_j \in \mathbb{X}\}$;
  $X \leftarrow \arg\max_{X_j | j \in R} \Delta f(X_j | \mathbb{X})$;
  $\mathbb{X} = \mathbb{X} \bigcup \{X\}$;
return $\mathbb{X}$;

the greatest marginal gain Δf found so far, we evaluate the actual Δf for the X_j, otherwise, we terminate the current iteration and selects the l-itemset that gives the greatest Δf so far. Initially, the upper bound for a l-itemset is computed at the first iteration against the empty itemset and is updated in subsequent iterations whenever the l-itemset's marginal gain is evaluated. For all l-itemsets that are not evaluated, the submodularity of f implies that their upper bounds remain unchanged because growing $\mathbb{X}$ never increases Δf of a l-itemset.

Lazy-forward only evaluates the l-itemsets whose upper bounds are larger than the best-known marginal gain. Even though the theoretical time complexity is the same as that of the straightforward implementation, the empirical running time is reduced due to fewer evaluations of marginal gains. However, the Lazy-forward implementation requires an extra priority-queue to store the upper bound of marginal gain for all l-itemsets, with $O(\binom{M}{l})$ space complexity.

Random-sampling. With Lazy-forward, Greedy algorithm needs to evaluate the marginal gains of all $\binom{M}{l}$ l-itemsets at the first iteration (before which all upper bounds are initialized as infinity), which is still time-consuming. To achieve better time efficiency, we adapt the sampling procedure from [25] to reduce the number of l-itemsets considered at each iteration. Specifically, at each iteration, we randomly sample $\frac{|\mathcal{X}|}{k} log(\frac{1}{\epsilon})$ l-itemsets from the set of all l-itemsets that have not been selected, and proceed with the sampled l-itemsets only, where $|\mathcal{X}|$ is the number of all l-itemsets and ϵ is in $(0, 1)$. According to [25], the approximation factor in Theorem 4 becomes $(1 - \frac{1}{e} - \epsilon)$. The Greedy algorithm with Random-sampling, called *Stochastic-Greedy algorithm* or simply *SGreedy*, is given in Algorithm 2. The sampling step is implemented by randomly sampling $\frac{|\mathcal{X}|}{k} log(\frac{1}{\epsilon})$ indexes for l-itemsets, i.e., R, and restricting the space for computing arg max to the l-itemsets with the indexes in R. This can be implemented as l nested loops, as for the straightforward implementation, except that it considers only the jth generated l-itemsets such that j is in R. In each iteration, this algorithm evaluates the marginal gain for no more than $\frac{|\mathcal{X}|}{k} log(\frac{1}{\epsilon})$ l-itemsets and the evaluation of marginal gain on each l-itemset takes $O(N \cdot l)$ time. Therefore, for k iterations, the time complexity is $O(\log(\frac{1}{\epsilon}) \cdot \binom{M}{l} \cdot N \cdot l)$. With the total sample size being no more than $|\mathcal{X}|$, SGreedy algorithm's time complexity is no more than that of the straightforward implementation. Similar to the straightforward implementation, this algorithm only takes the space $O(N \cdot M)$ for storing the input matrix.

To integrate Random-sampling with Lazy-forward, we note that the sets of l-itemsets at two adjacent iterations are sampled independently, and a straightforward integration requires building a different priority-queue structure at each iteration, which introduces considerable overhead (as

Table 3. The time and space complexities of different implementations of Greedy algorithm. In the case of $log(\frac{1}{\epsilon}) > k$, the time complexity of Greedy and SGreedy algorithms is the same since the sample size cannot be greater than the full size $|\mathcal{X}|$. ↓ means the empirical running time is reduced compared to without the Lazy-forward strategy, even if the theoretical time complexity remains the same.

Algorithms	Acceleration strategies	Space Complexity	Time Complexity	Expirical Running Time
Greedy algorithm	Straightforward	$O(N \cdot M)$	$O(k \cdot \binom{M}{l} \cdot N \cdot l)$	-
	Lazy-forward	$O(\binom{M}{l})$	$O(k \cdot \binom{M}{l} \cdot N \cdot l)$	↓
SGreedy algorithm	Random-sampling	$O(N \cdot M)$	$O(\log(\frac{1}{\epsilon}) \cdot \binom{M}{l} \cdot N \cdot l)$	-
	Random-sampling + Lazy-forward	$O(\binom{M}{l})$	$O(\log(\frac{1}{\epsilon}) \cdot \binom{M}{l} \cdot N \cdot l)$	↓

Algorithm 3: k-max algorithm

Input: A matrix $\mathcal{D}$, group number k, itemset size l, stop threshold θ.
Output: $(D_1, X_1), ..., (D_k, X_k)$
initialize distinct l-itemsets $X_1, ..., X_k$;
$iter\_increase = +\infty$;
$cur\_utility = 0$;
while *iter_increase* $> \theta$ **do**
  /* **Assignment Step**: induce $D_1, ..., D_k$ */
  for $i = 1;\ i \leq N;\ i = i + 1$ **do**
    assign row r_i in $\mathcal{D}$ to D_j such that $g(r_i, X_j)$ is maximum;
  /* **Update Step**: update $X_1, ..., X_k$ */
  if *some rows were assigned to a new group* **then**
    for $j = 1;\ j \leq k;\ j = j + 1$ **do**
      $X_j \leftarrow l$-itemset X with largest $\sum_{r_i \in D_j} g(r_i, X)$;
  $iter\_increase = f(\mathbb{X}) - cur\_utility$;
  $cur\_utility = f(\mathbb{X})$;
return $(D_1, X_1), ..., (D_k, X_k)$

verified empirically). To avoid this cost, we first check the upper bound of each sampled l-itemset and only evaluate its Δf if its upper bound is larger than the best-known marginal gain found. The worst case time complexity is the same as Random sample without Lazy-forward, but the empirical running time is reduced due to fewer evaluations of marginal gains. This algorithm needs $O(\binom{M}{l})$ space to store the upper bounds of all l-itemsets sampled in k iterations.

The time and space complexities of different implementations of Greedy algorithm are summarized in Table 3.

5 k-MAX ALGORITHM

The bottleneck of Greedy algorithms is identifying the best X_j at each step, which is time-consuming when there are many items, even with the Lazy-forward and Random-sampling techniques. In this section we present the second algorithm, named *k-max*, to address this issue. This name comes from the iterative refinement of the k groups, $(D_1, X_1), ..., (D_k, X_k)$. Given in Algorithm 3, k-max

starts with initializing k distinct l-itemsets, $X_1, ..., X_k$. In each iteration, the algorithm alternatively performs two steps:

- **Assignment Step** Given $X_1, ..., X_k$, this step assigns each row r_i to the group D_j such that $g(r_i, X_j)$ is maximized over $1 \leq j \leq k$. r_i stays in its current group D_j if $g(r_i, X_j)$ is already maximized.
- **Update Step** Given $D_1, ..., D_k$, this step updates each X_j to the l-itemset X such that $\sum_{r_i \in D_j} g(r_i, X)$ is maximized, in other words, X_j is the top-l items ranked by the sum of scores of the users in D_j on the item.

In each iteration, each step improves $X_1, ..., X_k$ or $D_1, ..., D_k$ assuming that the other is fixed. This iterative process continues until the increase of utility is smaller than the specified threshold θ, in which case each row r_i belongs to D_j that maximizes $g(r_i, X_j)$, that is, $D_1, ..., D_k$ is induced by $\mathbb{X}$ (i.e., $h(r_i, \mathbb{X}) = g(r_i, X_j)$).

The following theorem shows that, after the first iteration, $f(\mathbb{X})$ is no less than the utility of the standard top-l items $f^*(1, l)$.

Theorem 5. *For $\mathbb{X} = \{X_1, ..., X_k\}$ produced at the end of the first iteration, $f(\mathbb{X}) \geq f^*(1, l)$.*

Proof. $f^*(1, l)$ is $f(\{X\})$ for the top-l items X in the whole dataset. Let D_j be produced by Assignment Step in the first iteration and let X_j be produced by Update Step in the first iteration. Note that X_j is the set of top-l items in D_j, thus, $\sum_{r_i \in D_j} g(r_i, X_j) \geq \sum_{r_i \in D_j} g(r_i, X)$, and

$$\sum_j \sum_{r_i \in D_j} g(r_i, X_j) \geq \sum_j \sum_{r_i \in D_j} g(r_i, X)$$

The left-hand-side is $f(\mathbb{X})$ and the right-hand-side is $f^*(1, l)$. □

Initialization of $X_1, ..., X_k$. The simplest way of initializing $X_1, ..., X_k$ is randomly selecting these X_j's. Let *Random_Init* denote this random initialization. Another way is greedily selecting the X_j for $1 \leq j \leq k$ as the top-l items in the remaining data, initially $\mathcal{D}$. After selecting X_j, we remove the rows having X_j as its top-l items before selecting X_{j+1}. Let *Smart_Init* denote this greedy initialization. Unlike Random_Init, Smart_Init is deterministic. We will study the effect of these initializations experimentally.

Time complexity. Assignment Step takes $k \cdot l \cdot N$ time because it takes $k \cdot l$ time to find X_j that maximizes $g(r_i, X_j)$ for a row r_i. In Update Step, for each $1 \leq j \leq k$, the computation of X_j involves computing the top-l items in D_j, which takes time $O(|D_j| \cdot M)$ (with $|D_j|$ being the number of rows in D_j), therefore, Update Step takes time $O(N \cdot M)$. Ignoring the small constants l and k, the algorithm with τ iterations takes $O(\tau \cdot M \cdot N)$ time, which is a linear in the input matrix size $M \cdot N$. We will empirically evaluate the efficiency of this algorithm.

Convergence analysis. The algorithm always converges because the change of group membership is triggered by a positive increase of f and there is only a finite number of such increases. For a general matrix $\mathcal{D}$, the optimal utility is capped by the sum of top-l columns selected for each row. Let Θ denote this sum. Remember that θ is the threshold of increase for early stop. According to Theorem 5, after the first iteration, the increase of f is no more than $(\Theta - f^*(1, l))$. With each iteration increasing f by at least θ after the first iteration, the maximum number of iterations of k-max algorithm is no more than $\lceil (\Theta - f^*(1, l))/\theta \rceil + 1$. Typically, the increases of f in early iterations are much greater than θ, so the number of iterations needed is much fewer than $\lceil (\Theta - f^*(1, l))/\theta \rceil + 1$. We will evaluate empirically the number of iterations in the experiment section.

Table 4. A toy movie rating dataset

	m_1	m_2	m_3	m_4	m_5
r_1	2	0	3	3	2
r_2	4	1	4	3	1
r_3	2	4	4	0	1
r_4	3	2	1	0	1
r_5	2	4	4	3	1
r_6	3	2	3	3	3

Example 2. We illustrate the working of k-max algorithm using the toy movie rating data in Table 4, and $k = 3$ and $l = 2$. The optimal solution has the utility $f = 40$ given by the groups

$$(r_4,\ m_1m_2), (r_1r_2r_6,\ m_1m_3), (r_3r_5,\ m_2m_3)$$

Here m_1m_2 and $r_1r_2r_6$ are the short-hand for $\{m_1, m_2\}$ and $\{r_1, r_2, r_6\}$, and same for others. The standard top-l selection will find m_1m_3 with the utility $f = 35$ because these items have the largest score sums 16 and 19 on the whole dataset, respectively. Let us run k-max algorithm with an random initialization $X_1 = m_2m_4,\ X_2 = m_1m_3,\ X_3 = m_4m_5$.

First iteration: For r_2, the g function on X_1, X_2, X_3 returns $4, 8, 4$, respectively, so r_2 is assigned to D_2. After assigning all rows, the utility f becomes 36 with the groups

$$(r_5,\ m_2m_4), (r_1r_2r_3r_4r_6,\ m_1m_3), (\emptyset,\ m_4m_5)$$

In Update Step, X_1, X_2, X_3 are updated to maximize the total rating for their partitions of rows, i.e.$X_1 = m_2m_3,\ X_2 = m_1m_3,\ X_3 = m_4m_5$,

Second iteration: Reassign each row to the group that maximizes its total score. For example, r_3 is reassigned to m_2m_3 because it has total score 6 in m_1m_3 but has the total score 8 for m_2m_3. After Assignment Step, $f = 39$ given by

$$(r_3r_5,\ m_2m_3), (r_1r_2r_4r_6,\ m_1m_3), (\emptyset,\ m_4m_5)$$

After Update Step, all X_j's remain unchanged.

Third iteration: no row changes its group and the increase is 0, so the algorithm stops. The final utility $f = 39$, which is close to the optimal utility $f = 40$.

It is easy to verify that with the initialization $X_1 = m_1m_2, X_2 = m_2m_3, X_3 = m_4m_5$, we will get the following partitions with the utility $f = 40$ after two iterations,

$$(r_2r_4, m_1m_3), (r_3r_5, m_2m_3), (r_1r_6, m_3m_4)$$ □

6 EVALUATION

In this section, we conducted experiments to study the effectiveness of the proposed methods on three real life public datasets. Section 6.1 first introduces the experimental setup. Section 6.2 compares the solution of each algorithm with the optimal solution, and these experiments are conducted on small datasets because obtaining the optimal solution is time-consuming. Section 6.3 reports the experiments on summarizing users' preferences by comparing our summaries with those of the top-l selection. Section 6.4 studies the scalability of algorithms on a larger dataset. The codes are released on GitHub [2].

[2]https://github.com/wangyongjie-ntu/GUM

6.1 Experimental Setup

We consider the following setup to evaluate our proposed algorithms.

Evaluation metrics: We evaluate the average utility of the selected items, called *average item utility (AIU)*, mathematically defined as $\frac{f(\mathbb{X})}{k \times l}$ where $f(\mathbb{X})$ is defined in Equation (1). A larger AIU indicates a better summary of high utility items. For algorithms with random factors (i.e., random initialization in k-max algorithm and Random-sampling in SGreedy algorithm), we also report the *coefficient of variation* of AIU of multiple runs, i.e., $\frac{\sigma}{\mu}$ where σ and μ are standard deviation and mean, respectively. We also evaluate the *running time* of proposed methods. All experiments were implemented in Python on a Ubuntu server with Intel Xeon Silver 4114 Processor at 2.2 GHz, 187GB of memory, with matrix operations from the Numpy and Pandas libraries.

The proposed algorithms: The GUM problem has two parameters, k and l, that define the summary size $k \times l$. In practice, this size is bounded by the human effort required to read the l items for each of the k groups. For this reason, k and l cannot be too large, for example, $k \in [1, 10]$ and $l \in [1, 5]$. We study two proposed solutions to the GUM problem. The first is the Greedy algorithm presented in Section 4. **Greedy** denotes Greedy algorithm integrated with Lazy-forward strategy and **SGreedy**-ϵ denotes Greedy algorithm integrated with Lazy-forward and Random-sampling strategies (averaged over 10 runs). We set ϵ to 0.5 and 0.9 to explore different trade-offs between efficiency and approximation quality. The second algorithm is **k-max algorithm** introduced in Section 5. We consider three options of running this algorithm. **KMM** reports the best AIU of running k-max algorithm with 50 initializations by Random_Init and reports the total time of the 50 runs. **KMA** reports the average AIU and time of the 50 runs, which gives an estimated performance of a single initialization. **KMS** denotes running k-max algorithm with the greedy initialization Smart_Init. In our experiment, we set θ to 0 for k-max algorithm because it can run efficiently.

We compare our algorithms with several baseline algorithms. These baselines do not find a solution for the GUM problem, i.e., k groups of users and l items for each group, but they are the closest to our work.

Baselines: K-CPGC [10] denotes the max-sum submatrix algorithm, which finds k max-sum submatrices iteratively. **CoClust** [17] denotes the spectral co-clustering, which finds $k(\geq 2)$ nonoverlapping clusters. Both K-CPGC and CoClust do not constrain each submatrix to l columns. To convert their solutions for our GUM problem, if the column number of a submatrix is greater than l, we only keep top-l columns; otherwise, we make up l columns by adding columns that are outside of the submatrix and have the largest sum over the rows of this submatrix. Note that the submatrices found by K-CPGC may not cover all rows. We also consider **Random_Init** (averaged over 50 runs) and **Smart_Init**, i.e., the initialization methods of k-max algorithms as described in Section 5. Comparison with these initializations can tell the improvement by k-max algorithms. Finally, we include the **top-l selection** over the whole data set (i.e., $k = 1$), which has the utility of $f^*(1, l)$. The comparison with this solution tells the improved utility due to user partitioning. As discussed in Section 2, subgroup discovery and preference mining are not applicable to our problem.

Datasets: We experimented on three real-life datasets. **Titanic dataset** is a feature importance dataset of size 240×9 where each row denotes a correctly predicted survivor and each entry represents the importance of the corresponding feature value to the survival prediction. The detailed generation of Titanic dataset is shown in Section 6.2. **Netflix-Prize-Dataset** [5] contains about 100 million ratings in the scale from 1 to 5, given by 480, 189 users on 17, 770 movies. The absence of rating is filled by the value 0. Since many movies have very few ratings (i.e., the density of the full dataset is only 1.18%), we considered two datasets. **Netflix-Prize-17770** denotes the full dataset containing all movies, and **Netflix-Prize-200** denotes the denser dataset with the

Table 5. Statistics of processed datasets.

Dataset	#Rows	#Cols	Density	Scales
Titanic dataset	240	9	100%	-
Netflix-Prize-17770	480,189	17,770	1.18%	1,2,3,4,5
Netflix-Prize-200	480,189	200	25.66%	1,2,3,4,5
MovieLens-1B-200	2,210,078	200	2.10%	0, 1

density 25.66%, containing the 200 movies that have the most number of ratings. **MovieLens-1B** [4] is a synthetic dataset consisting of 2,197,225 users and 855,776 items with binarized ratings, that was expanded from the real-world rating dataset MovieLens-20M [18]. Due to the extremely small density of original dataset (e.g., half of items have less than 0.02% votes, the density of original dataset is 0.065%), we restricted to the 200 most voted items, and the resulting data has N=2, 210, 078 rows and M=200 columns with binarized ratings and density of 2.10%, denoting by **MovieLens-1B-200**. The statistics of processed datasets are summarized in Table 5.

6.2 Comparison with Optimal Solutions on Titanic Dataset

This experiment demonstrates an application of the GUM problem in explaining the prediction of a black-box model. We consider the Titanic data, previously used for Kaggle's Titanic machine learning competition, which describes the survival status (the class attribute) and other information of 891 passengers on Titanic, 342 survived and 549 died. We removed the superficial features passenger ID, ticket number, cabin number, and name, and the remaining features are given in Example 1. Our task is to explain the survival prediction made by a black-box model. To this end, we trained a four-layer multilayer perceptron (MLP) model $F(x)$ using the dataset to predict the surviving probability of a passenger x. The MLP model consists of 4 fully-connected layers of sizes (9, 20), (20, 20), (20, 10), and (10, 2) with 2 output units and ReLU activation function, with the binary cross entropy loss minimized by the SGD optimizer with learning rate 0.1. The accuracy and F1-score (for the surviving class) on the data set are 84.06% and 77.17%, respectively.

Next, we created a user-item matrix $\mathcal{D}$ containing 240 predicted correctly survivors of the full dataset, where there is a row r for each survivor x and there is a column for each feature. The entry for a feature x_i stores the Integrated Gradient (IG) [35] of $F(x)$ w.r.t. the feature x_i, as explained in Example 1. The computation of IG is based on a baseline point. We specify the baseline as the mean of all dead passengers that are predicted as dead. A solution to the GUM problem, $(D_1, X_1), \cdots, (D_k, X_k)$, would provide a high level explanation for the prediction of the survivors in $\mathcal{D}$, where each X_j contains the l features that have largest sum of IG for a group D_j.

Figure 1 compares AIU and running iterations. "Optimal" represents the optimal solution. The dashed line represents AIU of top-l items. First of all, Greedy and KMM have almost the same utility as Optimal; KMS and KMA are slightly lower than Optimal. While KMA is the result of a single initialization, it is about 2.60% lower than KMM that is the best result of 50 initializations. Also, the coefficient of variation (CV) of KMA is within 2.75%, suggesting a small variance due to random initialization. On the other hand, the single initialization offers most efficiency for dealing with large datasets, as shown in later experiments. Smart_Init achieves competitive results for small k but performs poorly for larger k. KMS always improves on Smart_Init. k-max algorithms consistently surpass the top-l selection, which is the result for $k = 1$. This is consistent with Theorem 5. The right figure shows the maximum number of iterations of the 50 runs for KMA, and the actual number of iteration required of KMM and KMS over various k. We observe that the largest iteration is 9, 6, 3 on Titanic dataset for KMA, KMM and KMS.

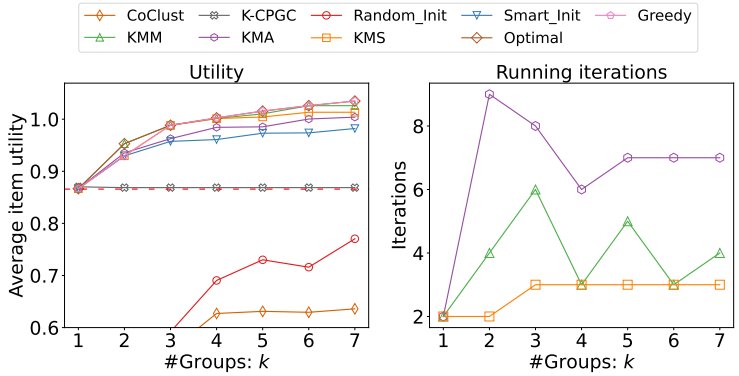

Fig. 1. Titanic dataset: AIU and running iterations for various k and $l = 3$. The red dashed line represents the utility for top-l selection. The coefficients of variation of Random_Init, KMA utility are within 74.92%, 2.75%.

However, in Figure 1, CoClust and Random_Init have the lowest utility, even lower than that of top-l selection. Due to nonoverlapping in columns, groups produced by CoClust cannot select the top important features simultaneously, resulting in the poor utility. K-CPGC selects 239 users from all 240 survivors in the first iteration. As we only keep the top-l columns if the column number of a submatrix is greater than l, K-CPGC achieves the similar utility as top-l selection.

Table 6 shows the three groups produced by Greedy and KMM algorithms with $k = 3$ and $l = 2$. As we can see, these features' importance in their groups is usually higher than on the whole dataset. Group 1 identifies Pclass and Sex (i.e., female in first-class cabin) are top-2 important features for 159 survivors' prediction; Group 2 selects Sex and Q (i.e., female and not embarked from Queenstown) are top-2 important features for 26 survivors' prediction. 93.5% passengers embarked from Queenstown are in third-class cabin; Sex and Age (i.e., young female) are top-2 important features for 55 survivors' prediction in Group 3. This group level summary provides an easier explanation on the prediction for the hundreds of survivors, compared to reading the l most important features for each survivor individually. On the other hand, top-l selection explains all survivors using the same top-l features (i.e., Pclass and Sex), which fails to distinguish the differences for different groups.

6.3 Summarizing Preferences of Netflix Movies

The second experiment was conducted on Netflix-Prize-Dataset to summarize users' preferences of movies. Our task is to answer "what kinds of movies people like (i.e., give a high rating)" using a compact summary with small k and l. We apply KMM with $k = 5$ and $l = 3$ to Netflix-Prize-17770. Table 7 shows the $k = 5$ groups generated and the $l = 3$ preferred movies for each group. "Pop" (Popularity) and "Avg rating" are the number of ratings and the average rating of selected movies in each group. For example, Group 3 summarizes the users who love the "Lord of the Rings" series, which share the common theme of "adventure" and "fantasy". Groups 2 and 4 share similar interests in "comedy" and "drama" movies, with Group 2 also loving "romance" movies. Groups 1 and 5 share the common theme of "adventure" and "action" while group 5 also likes "comedy" and "drama" movies. Such group-level preferences provide a concise summary of the entire user population, with most movies selected having both a large number of ratings and a large average rating in a group. In contrast, traditional ranking either by the number of ratings or by the average rating will have either a large number of ratings or a large average rating, but not both, and produces a single list of movies for all users, shown in the Table 8.

Table 6. Titanic dataset: the groups produced with $l = 2$ and $k = 3$ by Greedy and KMM. In this setting, both Greedy and KMM produce the same results. For example, group 1 has 159 survivals and selects Pclass and Sex as top-2 important features whose average item utility are 1.28 and 1.75 respectively. The second row denotes the average utility of an item on the whole population. The numbers in the parentheses represent the sizes of survivals.

	Pclass	Sex	Age	SibSp	Parch	Fare	C	Q	S
Whole Population (240)	0.82	1.53	0.25	-0.01	0.06	-0.02	0.18	0.06	-0.06
Group 1 (159)	1.28	1.75							
Group 2 (26)		1.80						0.98	
Group 3 (55)		0.79	0.89						

Table 7. The Netflix-Prize-17770: X_1, X_2, X_3, X_4, X_5 found by KMM with $l = 3$ and $k = 5$. E.g., the first group has size 172,843 and "Miss Congeniality" is one of the top-3 movies in X_1 and has 85,555 ratings with the average rating of 3.66 in this group.

Group ID	Movies	Genre	Pop	Avg Rating	Group Size
1	Miss Congeniality	Adventure|Comedy|Crime	85555	3.66	172843
	The Patriot	Action|Thriller	77183	4.07	
	Independence Day	Action|Adventure|Sci-Fi|Thriller	78110	4.07	
2	Pretty Woman	Comedy|Romance	67535	4.31	89646
	Forrest Gump	Comedy|Drama|Romance|War	65743	4.58	
	The Green Mile	Crime|Drama	67825	4.53	
3	Lord of the Rings: The Return of the King	Adventure|Fantasy	90008	4.79	101142
	Lord of the Rings: The Fellowship of the Ring	Adventure|Fantasy	91358	4.77	
	Lord of the Rings: The Two Towers	Adventure|Fantasy	93855	4.76	
4	The Royal Tenenbaums	Comedy|Drama	45443	3.93	60157
	American Beauty	Comedy|Drama	42848	4.38	
	Pulp Fiction	Comedy|Crime|Drama|Thriller	44142	4.45	
5	Pirates of the Caribbean: The Curse of ...	Action|Adventure|Comedy|Fantasy	33193	4.25	56401
	The Day After Tomorrow	Action|Adventure|Drama|Sci-Fi|Thriller	38438	3.71	
	Man on Fire	Action|Crime|Drama|Mystery|Thriller	35539	4.21	

Table 8. The Netflix-Prize-17770: the top-15 movies found by traditional ranking methods by popularity (the total number of ratings) or average rating.

Top-15 Movies by Popularity			Top-15 Movies by Average Rating		
Movies	Pop	Avg Rating	Movies	Avg Rating	Pop
Miss Congeniality	232944	3.36	Lord of the Rings: The Return of the King (Extended)	4.72	73335
Independence Day	216596	3.72	Lord of the Rings: The Fellowship of the Ring(Extended)	4.71	73422
The Patriot	200832	3.78	Lord of the Rings: The Two Towers(Extended)	4.70	74912
The Day After Tomorrow	196397	3.44	Lost: Season 1	4.67	7249
Pirates of the Caribbean: The Curse of Black Pearl	193941	4.15	Battlestar Galactica: Season 1	4.64	1747
Pretty Woman	193295	3.91	Fullmetal Alchemist	4.61	1633
Forrest Gump	181508	4.30	Tenchi Muyo! Ryo Ohki	4.60	89
The Green Mile	181426	4.31	Trailer Park Boys: Season 3	4.60	75
Con Air	178068	3.45	Trailer Park Boys: Season 4	4.60	25
Twister	177556	3.41	The Shawshank Redemption: Special Edition	4.59	139660
Sweet Home Alabama	176539	3.54	Veronica Mars: Season 1	4.59	1238
Armageddon	171991	3.58	Ghost in the Shell: Stand Alone Complex: 2nd Gig	4.59	220
The Rock	164792	3.77	The Simpsons: Season 6	4.58	8426
What Women Want	162597	3.43	Arrested Development: Season 2	4.58	6621
Bruce Almighty	160454	3.43	Inu-Yasha	4.55	1883

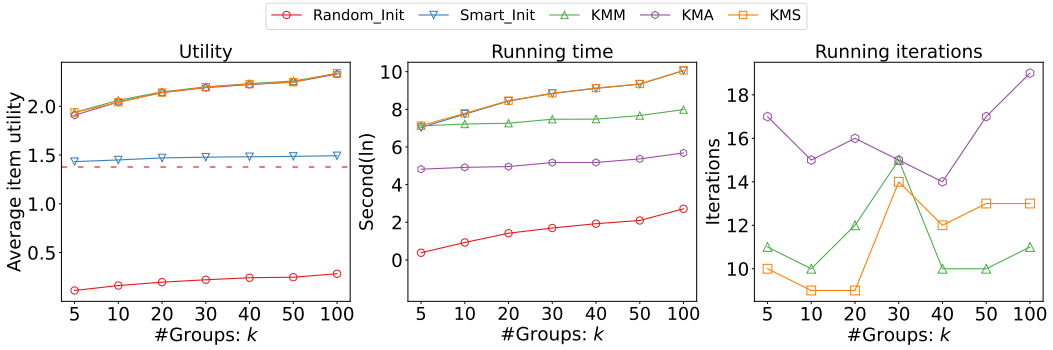

Fig. 2. Netflix-Prize-17770: The average item utility (AIU), running time (in ln scale), and running iterations for various k and $l = 30$. The red dash line represents the utility of top-l selection. CoClust, K-CPGC, Greedy and SGreedy-ϵ were omitted due to long running time. The coefficients of variation of Random_Init, KMA utility are within 6.96%, 1.21%.

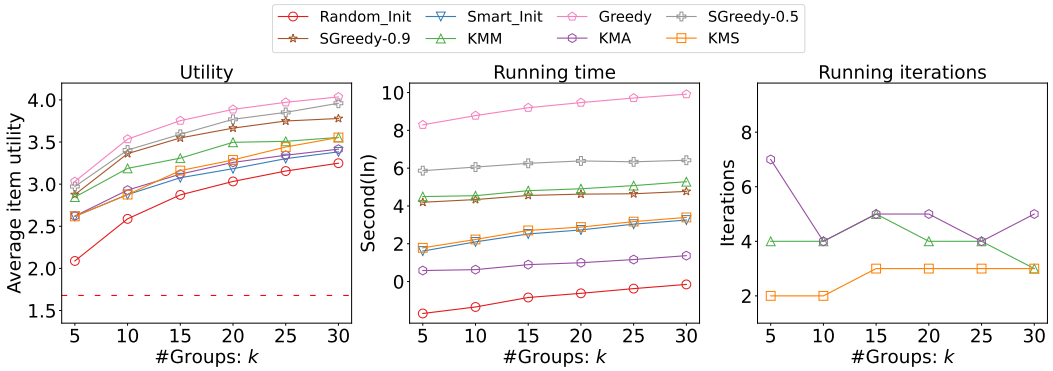

Fig. 3. Netflix-Prize-200: The average item utility (AIU), running time (in ln scale), and running iterations for various k and $l = 2$. The red dash line represents the utility of top-l selection. CoClust and K-CPGC cannot run within our limited time and are not reported. The coefficients of variation of AIU of Random_Init, SGreedy-0.5, SGreedy-0.9 and KMA and are within 6.90%, 1.14%, 1.40%, 5.71% respectively.

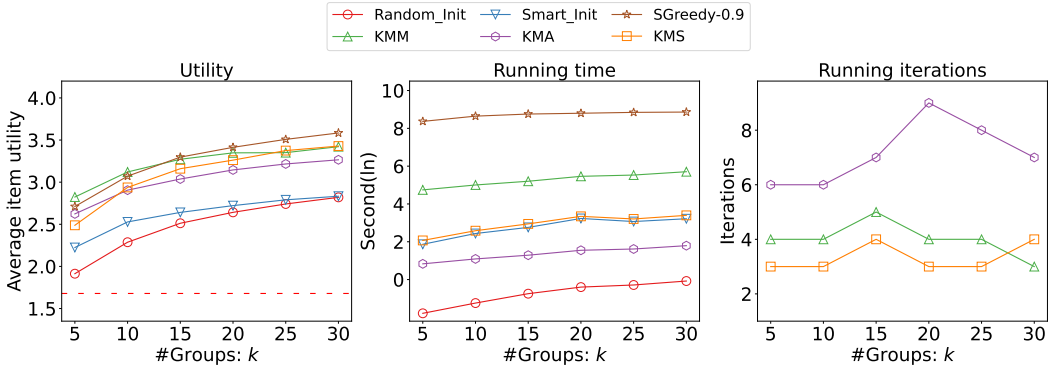

Fig. 4. Netflix-Prize-200: The average item utility (AIU), running time (in ln scale), and running iterations for various k and $l = 3$. The red dash line represents the utility of top-l selection. CoClust, K-CPGC, Greedy and SGreedy-0.5 were omitted due to long running time. The coefficients of variation of AIU of Random_Init, SGreedy-0.9 and KMA are within 4.95%, 1.22% and 3.86% respectively.

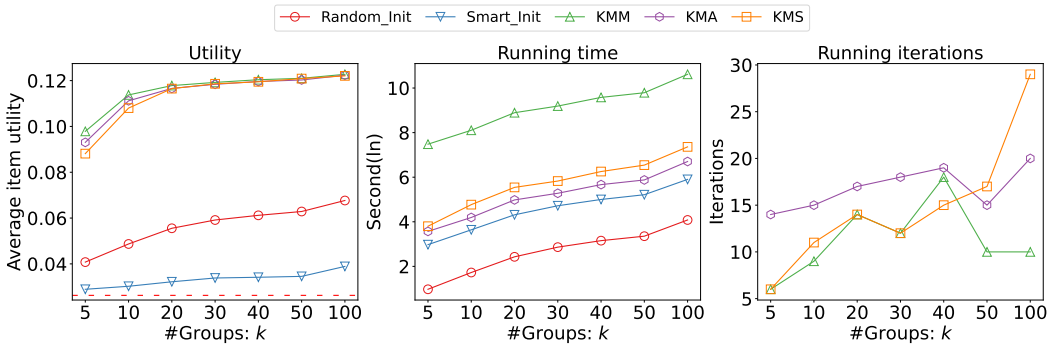

Fig. 5. MovieLens-1B-200: The average item utility (AIU), running time (in ln scale) and running iterations for various k and $l = 30$. The red dash line represents the utility of top-l selection. CoClust, K-CPGC, Greedy and SGreedy-ϵ are omitted due to long running time. The coefficients of variation of AIU of Random_Init, KMA are within 2.60%, 4.15%.

Figure 2 compares AIU, running time and running iterations for various k (x-axis) with $l = 30$ on Netflix-Prize-17770. Note that l is typically small because a large l would overwhelm the analyst. On this large dataset, only k-max algorithms, Random_Init, and Smart_Init can finish within our time limit (10 hours). In general, KMM achieves slightly better utility and KMA and KMS have similar utilities and surpass Smart_Init and Random_Init. Random_Init has the worst utility on this sparse dataset because most items have very few ratings. For running time, KMA is a big winner. Smart_Init takes significantly long time because of the intensive invokes of top-rank algorithm to find the next initialization greedily, which is computational bottleneck for the dataset with the larger number of movies. KMS runs Smart_Init in the initialization step, so is not faster than Smart_Init. Overall, KMA is a good trade-off between utility and efficiency on this large dataset. The right figure shows that the maximum iterations of 10 runs of KMA, actual iterations of KMM and KMS over various k. We can see that KMA, KMM and KMS stop within 19, 15 and 14 iterations.

Figures 3 and 4 report a similar study on the denser Netflix-Prize-200. K-CPGC and CoClust were not included because they cannot be run efficiently. Moreover, K-CPGC will return the entire matrix for a nonnegative matrix, which is not helpful for our purpose. Compared to the sparse Netflix-Prize-17770, all algorithms have much higher utility than top-l selection on this denser dataset, even for Random_Init, because the utility maximization due to group partitioning benefits more from a denser matrix. In general, Greedy and SGreedy-ϵ have a better utility than KMM, which has a better utility than KMS, which has a better utility than KMA. Smart_Init and Random_Init have a lower utility with Random_Init being the worst. Greedy and SGreedy-0.5 run slow for a larger k and l, and are not included for $l = 3$. For the running iterations, we can see KMA, KMM and KMS can converge quickly within 9, 5 and 4 iterations respectively on Netflix-Prize-200 dataset. Considering both utility and efficiency, KMA, KMM and KMS are preferred as they are less sensitive to larger k and l.

6.4 Scalability on MovieLens Data

The final experiment was conducted on the larger dataset MovieLens-1B-200 which has a larger number of rows than the above Netflix prize datasets. We use this dataset to evaluate the scalability of proposed algorithms.

Figure 5 shows the average item utility, running time and running iterations on the MovieLens-1B-200. Again, only k-max algorithms and Smart_Init and Random_Init can run efficiently. CoClust, K-CPGC, Greedy and SGreedy-ϵ cannot finish within a time limit (10 hours), therefore, are not

reported here. Consistent with previous experiments, KMA is most scalable and yet has a utility close to KMM. Smart_Init and Random_Init have very poor utility. The right figure demonstrates the maximum running iterations of KMA, the actual iterations of KMM and KMS over various k are within 20, 18, 29.

Due to the lower density of this dataset, there are several different findings. First, KMA, KMM, and KMS have a similar utility. This is because the small number of 1's in a row implies that there are few choices for the l-itemsets X_j, so different algorithms tend to have similar utility. Second, Smart_Init has a significantly lower utility. In each round of Smart_Init, X_j is selected as the top-l items in the remaining data and those rows having X_j as their top-l items are removed. However, few such rows were removed because few rows have X_j as their top-l items due to the low rating density. Consequently, the remaining data still contains most of rows in the next round, so the next X_{j+1} will be similar to X_j and the overall utility is similar to that of the top-l selection over the whole data.

6.5 Summary

While Greedy algorithm provides the approximation factor $(1 - \frac{1}{e})$ for a nonnegative matrix, the k-max algorithms (i.e., KMM, KMA, and KMS) are more efficient for large datasets. The small gap between KMM and KMA, as well as the small coefficient of variation of KMA suggests that KMA is a good trade-off between utility and efficiency. We recommend Greedy algorithms for small datasets and parameters k and l, and KMM for larger dataset and parameters k and l, and KMA for very large dataset and parameters k and l.

7 CONCLUSION

Summarizing a user-item matrix in terms of high utility items is of interest in many real-world applications. Existing techniques (e.g., clustering, subgroup discovery) fail to address this high utility requirement. We proposed a new summarization technique, named group utility maximization, and prove that the optimal solution is NP-hard. We proposed two algorithms, Greedy algorithm that adds one group at a time and provides a theoretical approximation bound for a nonnegative matrix, and the k-max algorithm that refines existing groups iteratively. Empirical studies show that Greedy algorithm provides a good utility whereas k-max algorithm is a good trade-off between utility and efficiency for dealing large datasets.

One future work is to explore more strategies to enhance the algorithm efficiency of both the Greedy and k-max algorithms. Possible ideas include representing a sparse user-item matrix with a compact representation and considering data compression techniques. Another future work is to study the approximation guarantee of the Greedy algorithm for a general matrix where an entry value can be both positive and negative. Last, we plan to apply our algorithms to a broader range of tasks such as one on the gene expression matrix, where each row/column denotes a gene/condition and a higher entry value denotes a higher expression level under a condition. We intend to discover highly expressed subsets of genes that are insightful to understand cellular processes.

REFERENCES

[1] Mohiuddin Ahmed. 2019. Data summarization: a survey. *Knowledge and Information Systems* 58, 2 (2019), 249–273.
[2] Sikder Tahsin Al-Amin and Carlos Ordonez. 2021. Efficient machine learning on data science languages with parallel data summarization. *Data & Knowledge Engineering* 136 (2021), 101930.
[3] David Arthur and Sergei Vassilvitskii. 2006. *k-means++: The advantages of careful seeding.* Technical Report. Stanford.
[4] Francois Belletti, Karthik Lakshmanan, Walid Krichene, Yi-Fan Chen, and John Anderson. 2019. Scalable realistic recommendation datasets through fractal expansions. *arXiv preprint arXiv:1901.08910* (2019).
[5] James Bennett, Stan Lanning, et al. 2007. The netflix prize. In *Proceedings of KDD cup and workshop*, Vol. 2007. Citeseer, 35.

[6] Jacob Bien and Robert Tibshirani. 2011. Prototype selection for interpretable classification. *The Annals of Applied Statistics* 5, 4 (2011), 2403–2424.

[7] Christian Borgelt. 2012. Frequent item set mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2, 6 (2012), 437–456.

[8] Vincent Branders, Guillaume Derval, Pierre Schaus, and Pierre Dupont. 2019. Mining a maximum weighted set of disjoint submatrices. In *International Conference on Discovery Science*. Springer, 18–28.

[9] Vincent Branders, Pierre Schaus, and Pierre Dupont. 2017. Combinatorial optimization algorithms to mine a sub-matrix of maximal sum. In *International Workshop on New Frontiers in Mining Complex Patterns*. Springer, 65–79.

[10] Branders, Vincent and Schaus, Pierre and Dupont, Pierre. 2019. Identifying gene-specific subgroups: an alternative to biclustering. *BMC Bioinformatics* 20, 1 (2019), 1–13.

[11] Church GM Cheng Y. 2000. Biclustering of expression data. In *Proc Int Conf Intell Syst Mol Biol*.

[12] Reuven Cohen and Liran Katzir. 2008. The generalized maximum coverage problem. *Inform. Process. Lett.* 108, 1 (2008), 15–22.

[13] Graham Cormode and Donatella Firmani. 2014. A Unifying Framework for l0-Sampling Algorithms. *Distrib. Parallel Databases* 32, 3 (sep 2014), 315–335. https://doi.org/10.1007/s10619-013-7131-9

[14] Marco De Gemmis, Leo Iaquinta, Pasquale Lops, Cataldo Musto, Fedelucio Narducci, and Giovanni Semeraro. 2009. Preference learning in recommender systems. *Preference Learning* 41 (2009), 41–55.

[15] Cláudio Rebelo de Sá, Wouter Duivesteijn, Carlos Soares, and Arno Knobbe. 2016. Exceptional preferences mining. In *International Conference on Discovery Science*. Springer, 3–18.

[16] Cláudio Rebelo de Sá, Wouter Duivesteijn, Carlos Soares, and Arno J. Knobbe. 2016. Exceptional Preferences Mining. In *DS*.

[17] Inderjit S. Dhillon. 2001. Co-Clustering Documents and Words Using Bipartite Spectral Graph Partitioning. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California) *(KDD '01)*. Association for Computing Machinery, New York, NY, USA, 269–274. https://doi.org/10.1145/502512.502550

[18] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article 19 (dec 2015), 19 pages. https://doi.org/10.1145/2827872

[19] Franciso Herrera, Cristóbal José Carmona, Pedro González, and María José Del Jesus. 2011. An overview on subgroup discovery: foundations and applications. *Knowledge and Information Systems* 29, 3 (2011), 495–525.

[20] ZR Hesabi, Zahir Tari, A Goscinski, Adil Fahad, Ibrahim Khalil, and Carlos Queiroz. 2015. Data summarization techniques for big data—a survey. In *Handbook on Data Centers*. Springer, 1109–1152.

[21] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)* 40, 4 (2008), 1–58.

[22] Dennis Leman, Ad Feelders, and Arno Knobbe. 2008. Exceptional model mining. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 1–16.

[23] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. 2007. Cost-Effective Outbreak Detection in Networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Jose, California, USA) *(KDD '07)*. Association for Computing Machinery, New York, NY, USA, 420–429. https://doi.org/10.1145/1281192.1281239

[24] S. C. Madeira and A. L. Oliveira. 2004. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 1, 1 (2004), 24–45.

[25] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. 2015. Lazier than lazy greedy. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[26] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming* 14, 1 (1978), 265–294.

[27] Chong-Wah Ngo and Feng Wang. 2009. Video Summarization.

[28] Carlos Ordonez, Naveen Mohanam, and Carlos Garcia-Alvarado. 2014. PCA for Large Data Sets with Parallel Data Summarization. *Distrib. Parallel Databases* 32, 3 (sep 2014), 377–403. https://doi.org/10.1007/s10619-013-7134-6

[29] Lance Parsons, Ehtesham Haque, and Huan Liu. 2004. Subspace Clustering for High Dimensional Data: A Review. *SIGKDD Explor. Newsl.* 6, 1 (jun 2004), 90–105. https://doi.org/10.1145/1007730.1007731

[30] Bidyut Kr. Patra and Sukumar Nandi. 2015. Effective Data Summarization for Hierarchical Clustering in Large Datasets. *Knowl. Inf. Syst.* 42, 1 (jan 2015), 1–20. https://doi.org/10.1007/s10115-013-0709-8

[31] Dan Pelleg and Andrew Moore. 1999. Accelerating Exact K-Means Algorithms with Geometric Reasoning. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Diego, California, USA) *(KDD '99)*. Association for Computing Machinery, New York, NY, USA, 277–281. https://doi.org/10.1145/312129.312248

[32] William G Ruesink. 1980. Introduction to sampling theory. In *Sampling Methods in Soybean Entomology*. Springer, 61–78.

[33] Kelvin Sim, Vivekanand Gopalkrishnan, Arthur Zimek, and Gao Cong. 2013. A survey on enhanced subspace clustering. *Data Mining and Knowledge Discovery* 26, 2 (2013), 332–397.

[34] Eric J. Stollnitz, Tony D. Derose, and David H. Salesin. 1996. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[35] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 3319–3328.

[36] Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, and Vipin Kumar. 2018. *Introduction to Data Mining (2nd Edition)* (2nd ed.). Pearson.

[37] Yongjie Wang, Ke Wang, Cheng Long, and Chunyan Miao. 2021. Summarizing User-Item Matrix By Group Utility Maximization. In *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1409–1414.

[38] Stefan Wrobel. 1997. An algorithm for multi-relational discovery of subgroups. In *European Symposium on Principles of Data Mining and Knowledge Discovery*. Springer, 78–87.

[39] Han Xu, Eric Martin, and Ashesh Mahidadia. 2015. Extractive Summarisation Based on Keyword Profile and Language Model. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Denver, Colorado, 123–132. https://doi.org/10.3115/v1/N15-1013