Project 4

ESE 545, Data Mining: Learning from Massive Datasets

Introduction

The project is aimed to process a massive dataset with programming techniques and then design and implement greedy algorithms to develop a recommender system to provide k recommendations of movies to advertise to users. The project consists of 4 parts and are solved by Python. All programs responding each question has been optimized in terms of running time and storage efficiency.

Q1.1 Dataset Reading and Processing Data

The original file contains user information file, movie information file and ratings file. As required by the question 1.1, a proper data representation need to be designed to store the ratings for m movies from n users. In this question, it is decided to use a matrix with each row representing ratings from a user to all m movies and each column representing ratings for a specific movie from all n users. The read-in and transformation process is descried below.

i. Read in the user information file and movie file to obtain the total number of users (m) and movies (n), which decides the size of the rating matrix.
ii. Initialize a matrix consisting of zeros in the size of m×n. Read in the ratings file. Each line of file streams of ratings file records the user ID, movie ID and the corresponding rating separated by "::".
iii. Iterate each line of ratings stream, record the user ID, movie ID and the rating to a list. Access the list and update the values in rating matrix with (user ID-1) as row number and (movie ID-1) as column number and rating score as the value at this position. After iterations return the rating matrix.

Q1.2 Monotonicity and Submodularity

An objective function *F(A)* is derived to measure how much the users favors the chosen subset based on the rating information. For this project the corresponding *F(A)* is calculated as follows:

$$F(A) = \frac{1}{n}\sum_{i=1}^{n} max_{j \in A} r_{i,j}$$

where $r_{i,j}$ denotes the rating that user *i* assigns to movie *j*. Basically the function is the average of the maximum ratings among the movies in the subset for each user.

In order to simplify the algorithm in later parts, we need to prove that the objective function is both monotone and submodular.

Prove Monotonicity

As illustrate in the following table, assuming subset A has a movies (a<m) that has been rated by all n users and subset B has these a movies plus b-a new movies from the whole movies set (b<m). This way subset $A \subseteq B$

A B

Ratings	Movie 1	Movie 2	…	Movie a	Movie a+1	Movie a+2	…	Movie b	…
User 1			$r_{1_{maxA}}$				$r_{1_{\max(B-A)}}$		
User 2			$r_{2_{maxA}}$				$r_{2_{max(B-A)}}$		
…									
User n			$r_{n_{maxA}}$				$r_{n_{max(B-A)}}$		

Assuming that in subset A, for each user, the ratings of his favorite movie (highest rating) is

$$r_{i_{maxA}} = r_{i,j}, \qquad j \in A$$

Then in subset B, considering the subset B-A, which means movie $j \in B$ and $j \notin B \cap A$, for each user, the ratings of his favorite movie (highest rating) over this particular set is

$$r_{i_{maxB-A}} = r_{i,j}, \qquad j \in B - A$$

Then the comparison of these two maximum values leads the result into two cases:

i) For every user the maximum rating in subset A is larger than or equal to that in subset B-A

$$r_{i_{maxA}} \geq r_{i_{maxB-A}}$$

for any i=1,2…n, which means for subset B, the maximum rating from each user is still $r_{i_{maxA}}$.

Then

$$F(B) = \frac{1}{n}\sum_{i=1}^{n} max_{j \in A} r_{i,j} = F(A)$$

ii) For at least one user, there exist a larger than maximum ratings from subset B-A.

$$r_{i_{maxA}} < r_{i_{maxB-A}}$$

Then

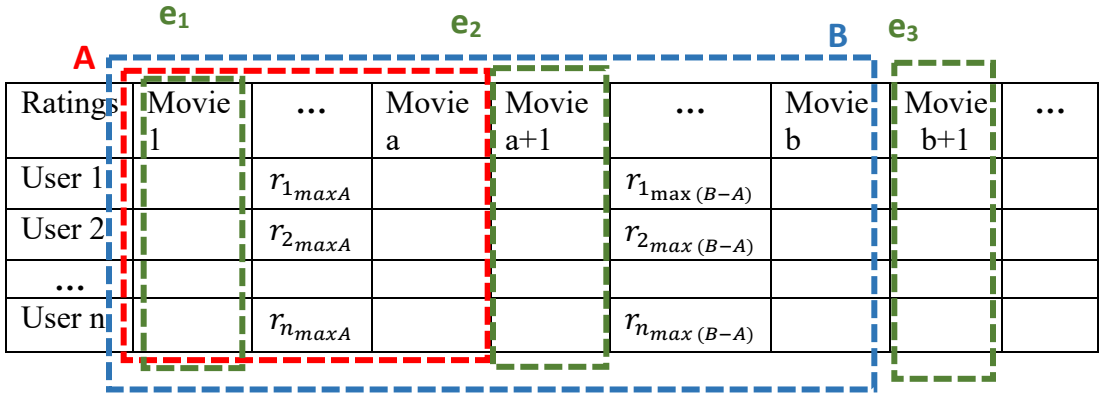
$$\sum_{i=1}^{n} max_{j \in B} r_{i,j} > \sum_{i=1}^{n} max_{j \in A} r_{i,j}$$

$$F(B) = \frac{1}{n}\sum_{i=1}^{n} max_{j\in B} r_{i,j} > F(A)$$

In summary, if the subset $A \subseteq B$, then $F(B) \geq F(A)$. Therefore the objective function for this problem is monotone.

Prove Submodularity

In order to prove that the objective function is also submodular, again, we re-construct two subsets A and B such that $A \subseteq B$. Additionally, a specific element e is added to both A and B as illustrated below.



Ratings	Movie 1	…	Movie a	Movie a+1	…	Movie b	Movie b+1	…
User 1		$r_{1_{maxA}}$			$r_{1_{\max(B-A)}}$			
User 2		$r_{2_{maxA}}$			$r_{2_{max(B-A)}}$			
…								
User n		$r_{n_{maxA}}$			$r_{n_{max(B-A)}}$			

If we consider an additional element which, in this case, is another movie, added into A and B, there are at most different cases as marked above.

i) $e_1 \in A$, which means $A \cup \{e_1\} = A,\ B \cup \{e_1\} = B$

Then $F(A \cup \{e_1\}) = F(A),\ F(B \cup \{e_1\}) = F(B)$

$$F(A \cup \{e_1\}) - F(A) = 0 = F(B \cup \{e_1\}) - F(B)$$

ii) $e_2 \notin A, e_2 \in B$ which means $A \subseteq A \cup \{e_2\},\ B \cup \{e_2\} = B$

Since $A \subseteq A \cup \{e_1\}$, due to monotonicity as proved above, $F(A \cup \{e_2\}) - F(A) \geq 0$

As $B \cup \{e_2\} = B, F(B \cup \{e_1\}) = F(B)$, therefore

$$F(A \cup \{e_2\}) - F(A) \geq 0 = F(B \cup \{e_2\}) - F(B)$$

iii) $e_3 \notin A, e_3 \notin B$

In this case, assuming that for movie e_3, the rating from each user is r_{ie3}

If for every user, $r_{ie3} < r_{i_{maxA}}$ $for\ any\ i = 1,2,\dots,n$,

Then the maximum ratings from each user for subset A and B remain the same, which means,

$$F(A \cup \{e_1\}) - F(A) = 0 = F(B \cup \{e_1\}) - F(B)$$

If there exists a user i such that $r_{i_{maxB}} > r_{ie3} \geq r_{i_{maxA}}$

Then the $F(B \cup \{e_1\}) - F(B)$ is still zero because the maximum ratings from each user for subset B remain the same. But $F(A \cup \{e_1\}) \geq F(A)$,which means,

$$F(A \cup \{e_1\}) - F(A) \geq 0 = \ F(B \cup \{e_1\}) - F(B)$$

If there exists a user i such that $r_{ie3} \geq r_{i_{maxB}} \geq r_{i_{maxA}}$,

then note that since $r_{i_{maxB}} \geq r_{i_{maxA}}$, the update

$$\Delta r_{ie3_A} = r_{ie3} - r_{i_{maxA}} \geq \Delta r_{ie3_B} = r_{ie3} - r_{i_{maxB}}$$

which further leads to

$$\sum_{i=1}^{n} max_{j \in A \cup \{e_1\}} r_{i,j} - \sum_{i=1}^{n} max_{j \in A} r_{i,j} \geq \sum_{i=1}^{n} max_{j \in B \cup \{e_1\}} r_{i,j} - \sum_{i=1}^{n} max_{j \in B} r_{i,j}$$

Therefore,

$$F(A \cup \{e_1\}) - F(A) \geq \ F(B \cup \{e_1\}) - F(B)$$

In summary, considering the above three cases,

$$F(A \cup \{e_1\}) - F(A) \geq \ F(B \cup \{e_1\}) - F(B)$$

Therefore, the objective function *F(A)* is submodular.

Q1.3 Greedy Algorithm

As required by the question, to maximize *F(A)* regarding to different value of k, Greedy Algorithm is implemented to expand subset *A* element by element. The detail steps are described as follows.

i. Initialize A matrix in the size of 1× n consisting of zeros. Note the original subset is supposed to be an empty set. But here the 1× n vector only represents the maximum rating from each user for all movies in the subset and is set to zero ready for replacement by higher value. Also, an empty list *F_record* is initialized to record the values of *F(A)* at different k
ii. Transpose the rating matrix to n × m. Compare each line in rating matrix with the above A vector directly using *numpy. maximum ()* which stores the larger element in A and each line of rating matrix. The resultant matrix is still in n × m size and denoted as *max_matrix*. The physical meaning of row j in such a matrix is the maximum ratings from each user in new subset if movie j was added into the subset.
iii. Then calculate the mean value for each line of *max_matrix* and find the maximum one as new *F(A).* Since the question does not require us to display the chosen movies set, instead of actually adding the chosen movie into the subset, we only update A vector to record the "contribution" the new movie has made to maximum ratings. This is done by replace original A with the line in *max_matrix* that has the highest mean value.

iv. Repeat step ii and iii for k times. Here k is set to 50. For every 10 times, append the *F(A)* in that iteration into *F_record* list. Then we will have objective values of the greedy algorithm versus k for k = 10, 20, 30, 40, 50. Plot the figure.

After running the program, the objective values of the greedy algorithm versus k for k = 10, 20, 30, 40, 50 is obtained as shown below.

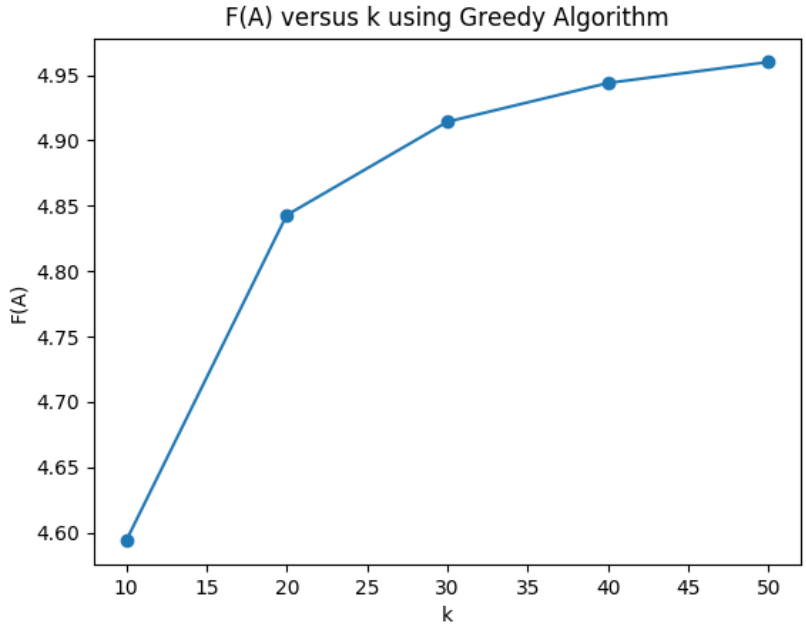


Figure1. Objective values versus k using Greedy Algorithm

From the plot above, it could be noticed that the objective value increases as the size of subset k rises, which is consistent with the monotonicity of the objective function proved in Q1.2.

Q1.4 Lazy Greedy Algorithm

The complexity of original Greedy Algorithm limits the efficiency of the recommender system. For further improvements, Lazy Greedy Algorithm is applied to process the movie sets and raise recommendations. The algorithm is based on the fact that in each iteration, the second greatest candidate (movie) is highly likely to be the greatest candidate for the next iteration. Before implementation, a new variable is introduced into the system $\Delta(e|A_i)$, which is the profit for *F(A)* after e is appended to the subset. The recommender system based on this algorithm is implemented following the listed steps below:

i. Initialize the subset maximums, mean vector and delta list (i=1). Create a mean vector to record the mean values for each line in rating matrix (or updated *max_matrix* when k>1). Similar to

our solution in Q1.3, rather than set A to be the actual subset, we set A to be a 1× n vector representing the maximum rating from each user for all movies in current subset. Therefore, when k=1, A is set to be the line in ratings matrix with largest value in *mean_vector*. Consequently, the current object value denoted as *F_max* in coding is exactly the greatest value in mean vector.

ii. Now the greatest element in *mean_vector* is added into the target subset and should not be taken into consideration in future. Therefore, we set the greatest element in *mean_vector* to zero before sorting in descending order. Then delta list denoted as *sorted_deltas* is just the sorted *mean_vector* in descending order. Note that this way the top one in *sorted_deltas* is actually from the second candidate (second one from third candidate etc.).

iii. For i>1, in each iteration, firstly calculate new delta for top one movie in *sorted_deltas* by substracting the previous *F_max* from new objective value. Compare the new delta value with second element in *sorted_deltas*. If the former is greater or equal to the latter, then according to submodularity and monotonicity, the profits from this movie is the greater than the rest. Therefore, update the new maximum vector *A*, *F_max* and mean vector accordingly. The *sorted_deltas* list remains the same, but the first element is deleted since that movie has just been added to target subset. However, if the new delta is less than the second element in *sorted_deltas,* we need to repeat Greedy Algorithm in Q1.3 and search over the whole movie set to update *mean_vector*, the subset maximum vector *A* and re-order the delta list.

iv. Repeat step iii for (k-1) times and record the corresponding *F_max* and running time every 10 rounds.

Again, for k=50, the resultant objective values versus k is plotted below.

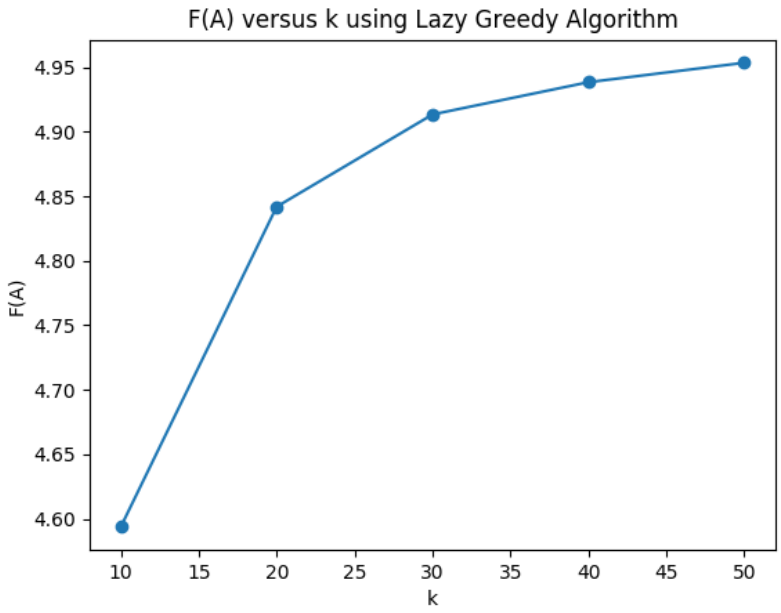


Figure2. Objective values versus k using Lazy Greedy Algorithm

From Figure2, the objective values are the same as the ones in figure2, which means Lazy Greedy Algorithm leads to the same result.

In terms of the efficiency of two algorithm, the running time for both algorithms is plotted below.

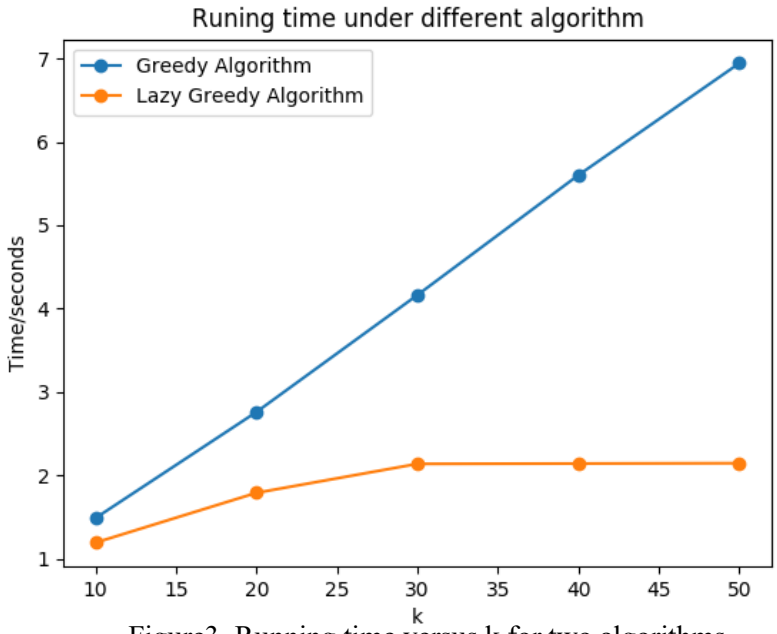


Figure3. Running time versus k for two algorithms

Obviously, Lazy Greedy Algorithm is much more efficient.