Project 1

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 algorithms to search for similar pairs as required. The project consists of 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 Transform

As required by the question, the original dataset, consisting of records of ratings for movies, need to be transformed into an M×N matrix, where M is the number of movies and N is the number of users. If the ratings from the user is higher than 2.5, then '1' ('0' otherwise) should appear at the corresponding location in the matrix. The task is achieved by the following steps:

i. Read from csv file and save values in each column (user, movie and rating) in three separate lists. Simplify the rating list by generate a Boolean list with *True* representing ratings over 2.5 and *False* for ratings less than or equal to 2.5.
ii. Match users/movie IDs with the corresponding column/row number in the giant matrix. Since user ID in the original file are consecutive integers, it is quite obvious that the column number is simply (user ID - 1). While for each movie's row number, a dictionary is implemented to match each movie ID with its row number. (i.e. the indexes of the dictionary are the movie IDs and the values are row numbers starting from 0)
iii. Create an all-zero numpy matrix in the required size (M×N). Then iterate through the new rating list, if true, then access the previous user lists and movies lists and acquire the corresponding user ID and movie ID. Next compute the row/column number from IDs as described in step ii. Finally change the value at this location in the matrix to '1'.

Q1.2 Jaccard Similarity for 10000 User Pair

The calculation and display of Jaccard Similarity for 10,000 randomly chosen user pairs is completed with following steps

i. Randomly select distinct 10,000 samples of user pairs
ii. Iterate through all user pairs and compute Jaccard Similarity for those two columns in the data matrix
iii. Calculate the average value of the entire similarity list. Then sort the list to obtain 10 largest similarities.
iv. Plot the histogram of Jaccard Similarity with x axis representing the similarity range and y axis representing the frequencies (number of users within each range)

The result for Q1.2 are obtained as below:

i. Average similarity= 0.0413169115884

ii. Ten largest Jaccard similarities among them are:
[0.39416058394160586, 0.39423076923076922, 0.41818181818181815,
0.42857142857142855, 0.47499999999999998, 0.47999999999999998,
0.48484848484848486, 0.49090909090909091, 0.5757575757575758
0.57692307692307687]

iii. The histogram of similarity:

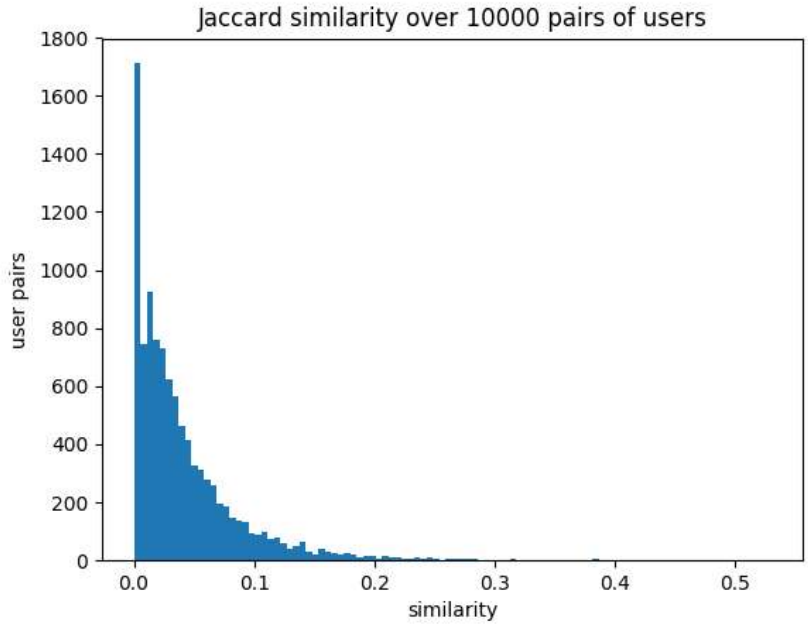

Figure1. Histogram of Jaccard similarity for 10000 user pairs

Q1.3 Efficient Data Storage

In order to release the space occupied by massive number of zeros in previous matrix, a more efficient way of data storage is designed that only the movies favorited (rating >2.5) by users are recorded in each user set as displayed below. The format could be illustrated as below:

[[moive1, movie 3, movie 7…], ##for user 1

[movie 3, movie 56 …], ##for user 2

…

[] ##for a certain user who does not like any movies

…

[movie 4, movie 256…, movie M]] ##for user N

Note that since the maximum movie ID is much larger than total number of rated movies. Therefore, same with the row number of the matrix in Q1.1, each favorited movie is represented using the "successive number" of all rated movies after sorting by the value of IDs. This representation is suitable for applying hash functions in Q1.4 and is also more memory efficient.

The detailed process are as follows:

i. Repeat step i and ii in Q1.1
ii. Initialize the user information list to hold all user's movie list
iii. Again, iterate through the Boolean rating list and obtain the corresponding user ID and movie ID if *True.* Then add the successive number of that movie to the user list with this user ID

Q1.4 Search for Similar Pairs

The question requires us to pick out all user pairs that have a Jaccard Similarity higher than 0.65. To efficiently searching for qualified user pairs, a combining technique of Minhash and LSH is applied to search for candidate pairs first and then compute the actual Jaccard similarities among these pairs and delete the unqualified pairs.

Before programming the band number (b) and row number(r) need to be decided for LSH applied on signature matrix. It is known that the probability for two users to become candidate pair is:

$$Pr = 1 - (1 - s^r)^b$$

where s is the Jaccard similarity. The function has the form of S-curve and the threshold value decided by b and r should equal to the required similarity, 0.65. A rough approximation of threshold value is $(\frac{1}{b})^{1/r}$. Considering the accuracy of this approach, b and r are chosen to be 100 and 10. Then the plot for above function will be:

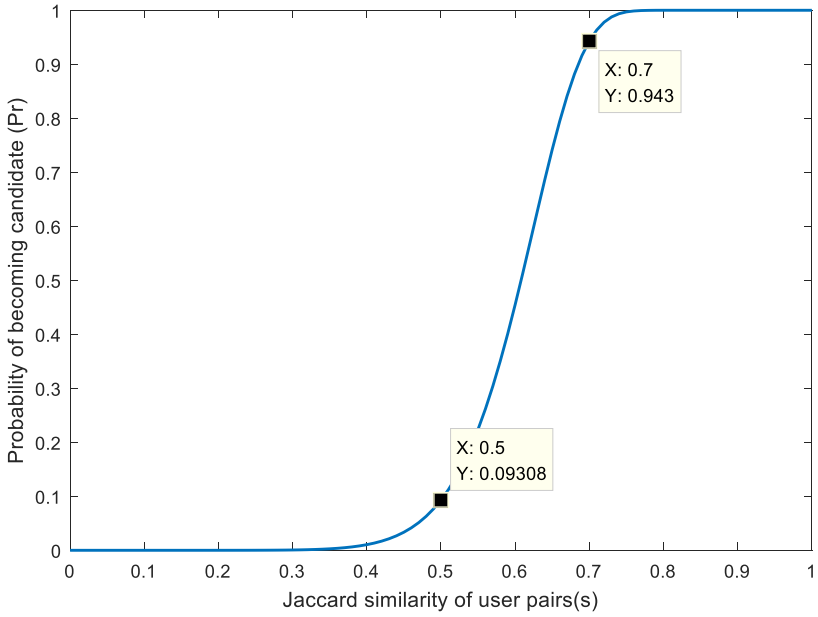

Figure2.Plot hitting probability over Jaccard similarity

As marked in the figure, the probability at s=0.7 is over 90% and the probability at s=0.5 is less than 10%. The threshold value, which is at the steepest slope, is around 0.65 hence satisfies the requirement.

The detailed steps for the whole algorithm are listed below:

i. Make use of the result of Q1.3 and obtain the user information list.
ii. To build hash functions in the form of $ax + b \; mod \; P$, two coefficient lists are generated for a and b to choose from. Each coefficient list consists of 1000 unrepeated integers. The value of P should be a prime number and is chosen to be 260171.
iii. Iterate through every user list in user information list. Within each loop for one user, calculate and record the minimum after applying a certain hash function on its favorited movies. And iterate 1000 hash functions and repeat the process. The results for each user are recorded in a 1×1000 signature list. The signature list is further added into a signature matrix.

iv. Define a hash function to vectors consisting of those 10 signatures as below:

$$h_v = \sum_{i=1}^{10} h_{Vi}\,, h_{Vi} = a_i x + b_i \; mod \; P$$

Transpose the obtained signature matrix so that rows represent for hash functions and columns for users. Before dividing into bands, compute the result of h_{Vi} function for each user using vector multiplication. Divide the rows into 100 bands so each band has responses from users to 10 signatures. Then calculate the sum of result in each band. If the results of multiple users are equal then hash the combination into one bucket. Iterate through all bands and repeat.

v. Now each bucket represents a user group in which users' signature are identical in for a certain band. Remove duplicates from all the buckets, which is possible because two users can be identical in multiple bands. Within each bucket, list all possible combinations of user pair. (For example, a bucket of (1,3,4) has three candidate pairs). Check if anyone in each pair has zero-length list, which means this user does not like any movies, then ignore those candidate pairs. Calculate the actual similarity for each of the rest candidate pairs and collect the ones that does has a similarity over 0.65. Finally remove the duplicates from actual candidate pairs.
vi. Export actual candidate pairs to a csv file.

Following the procedure above, a similar user pair list is obtained with a size of 94313. Note that the result may vary every time we run the program since the hash function are generated randomly.

Q1.5 Search for Nearest User

The question asks us to find the nearest neighbors for a given user ID, which can be solved by iterations of LSH function based on the signature matrix obtained in Q1.4. With a fixed signature matrix (1000 signatures), each iteration of LSH function changes the threshold value by updating values of b and r. A simple illustration is shown by the plot.

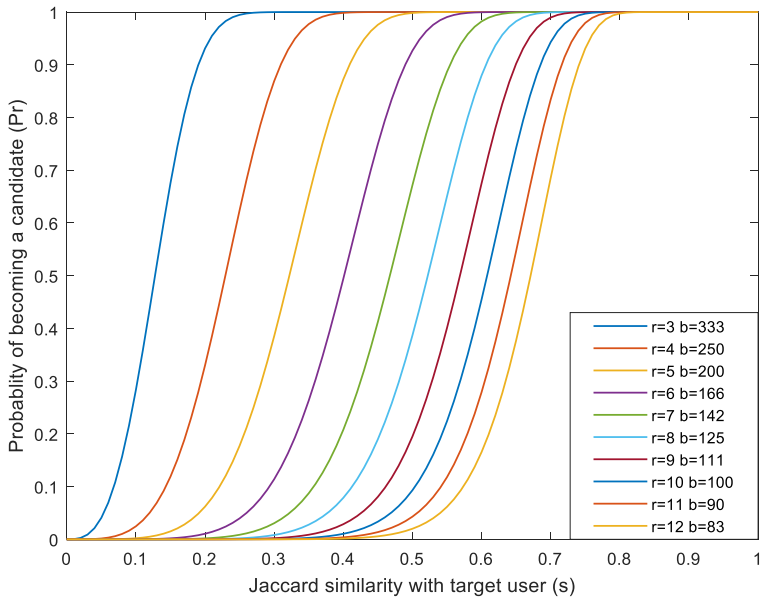

Figure3.Plot of probability over Jaccard similarity

The above figure displays 10 iterations of LSH function with a fixed number of br=1000. With the increment of r, b decreases and the threshold similarity increases, which implies that other users need to be more similar to target user to be selected as candidate. This way the group of candidate neighbors gradually shrinks and leave us the nearest ones.

The detailed process is described below:

i. Make use of the result of Q1.4 and obtain signature matrix generated from 1000 Minhash functions
ii. Starting with r=3 and b=333, which gives a threshold similarity of approx. 0.15. Then apply LSH and banding technique on the signature matrix in the same way as in Q1.4. The result of vector hash function is compared again. But instead of picking out all pairs that share the same result, pick out the users that have the same result with the target user (query point) and store them in the candidate list.

iii. If number of candidate neighbors is still over 10, then increase r by 1 and start a new round of LSH as suggested above. Record candidate list for each round. Iterate until the candidate neighbors are less than 10.
iv. Give a candidate list of users (less than 10), iterate to calculate their actual similarity with the target user. Compare the result with the current threshold (approximated as $(\frac{1}{b})^{1/r}$). And collect the users that are truly that similar with the target in a new list. If the number of actual neighbors become zero, access the candidate list in previous round with a lower threshold and select actual neighbors again. Then calculate the maximum similarity among this actual neighbor group and select the users that has a similarity same with the maximum value. That will be the answer for Q1.5.

The result for Q1.5 varies with different query point. From one certain trial with target user Id=7, the most nearest user is user 116446