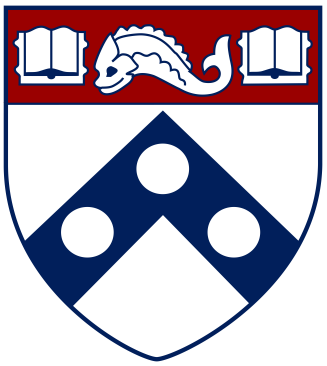
by -

ESE 545: Data Mining

Similarity of Netflix Users

By

RENZHI HUANG & YANCI ZHANG



SCHOOL OF ENGINEERING AND APPLIED SCIENCE
UNIVERSITY OF PENNSYLVANIA

SEPTEMBER 2018

Code Structure

To run our code through problem 1-5, just run: **run.py**. There are comments denoting the section for each problem.

Especially, the function: **get_queried_user()** is the function for problem 5.

Problem 1 and 3

The most straightforward way to store the data is using a array of size movie x user. Since the array will be very large, it will cost a lot of memory to do computations.

Therefore we can take advantage of the feature of this array, that is the array will be very sparse. We can use a scipy sparse array or a dictionary to store the liked-movies index (which is the index at which the previous array is 1) per user. We implemented our application with the latter approach. With this approach, it is faster to calculate minHash since only valid (value = 1) movies will be accessed. Also it will be intuitively easy to calculate the Jaccard Simularity in this case.

Problem 2

By calculating Jaccard distance of a random set of pairs with size 10,000, the average distance is 0.98, indicating users are generally so different from each other. The minimum distance varies due to different sampling (in most case it is 0.5). The histogram is shown in Figure 1.

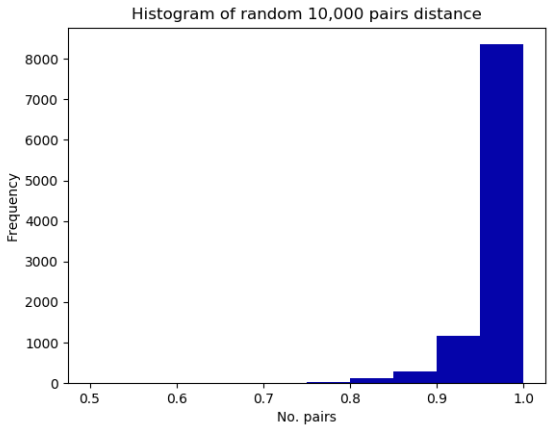


Figure 1: Histogram of random 10,000 pairs distance, bin size is 10

Problem 4

The parameters are used as followings:

```
number of hash = 161
prime number used in minhash = 4507
threshold = 0.65
length per band (r) = 7
prime number used in bucket = 4523
```

The overall target will be finding closest users with fairly low False Negative rate and comparable computational speed. There are several concerns and trade offs for selecting these parameters in number of hashes(m), band size(b) and number of elements per band(r).

For the choice of m, there are some considerations:
1. In order to minimize False Negative, we need to pick a m as large as possible so that $(1-s)^m$ would be small.
2. In order to balance FN and FP (get lower FN while FP won't be too large), we would like to have the slope of the function as steep as possible, so that areas on the left side under the curve and area on the right side of threshold above the curve would be minimized. m should be chosen relatively large.
3. m large means computation intensive. For the perspective of minimum computation, m should be limited to an optimum level that has the performance but does not use too much redundancy.

The following steps are taken to finalize parameters. In Figure 2, we have chosen r to make the graph balance both FN and FP in order to select m. m = 1000 and r = 10 has been tried, it could find 1.95 million pairs but cost twice as much memory and about 1.5 hours, while m=161 finds 1.90 million pairs but only take 12 mins. According to Figure 3 and 4, the most balanced parameters comparison shows that not using m = 161 hashes is very close to the performance of m = 256, so m = 161 is chosen.

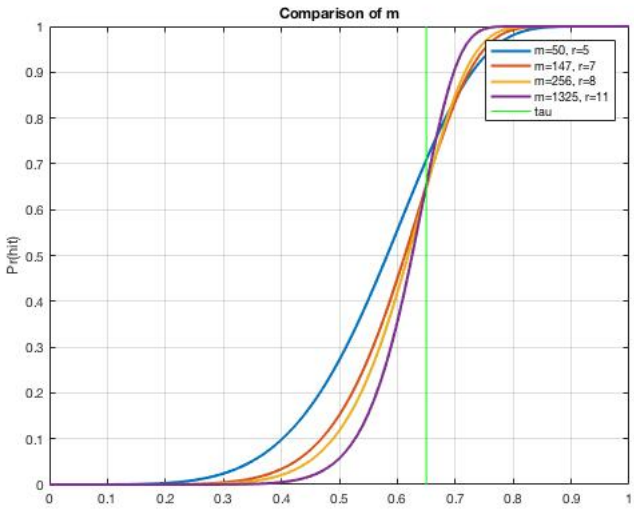


Figure 2: Comparison of m

In terms of r, a larger value would reduce the misses, which are False Negative, but in the cost of bringing up more False Positive. To verify each pair column directly, if we have too much FP, it would significantly increase our computation. By experiment, if r is chosen to be 7, time to find 1.88 million pairs would take 250 seconds, while r is chosen to be 5, time to find million pairs would take 1300 seconds. It is not obvious misses, but almost three time computational time.

Therefore, our parameter is chosen to be m = 161 and r = 7.

The steps to find the final pairs are as following:
1. Preprocess the txt file to a dictionary, in which the key is a user, and the value is a list containing all the liked-movies of this user.
2. Calculate the signature matrix for all the users.

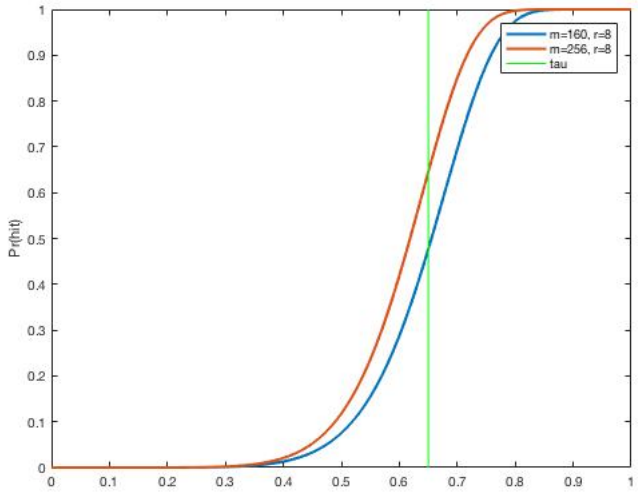


Figure 3: Comparison of m =160 and m=256

3. Hash the bands of signature matrix respectively. The users who are in the same bucket for at least one band are considered as candidate pairs.
4. Calculate the signature matrix similarity for the candidate pairs and delete those whose similarity is below 0.65.
5. Calculate the Jaccard Distance using the dictionary mentioned in step 1 and delete those whose similarity is below 0.65.
6. Write the final pairs into a csv file.

Problem 5

Our present approach is taking in a input list containing the liked-movies for a queried user and returning his nearest neighbor whose Jaccard distance with him is smallest.

The main idea takes advantage of the pre-calculated buckets for the each band of the signature matrix.
1. Calculate the min-hash signature using the same min-hash functions (total number = m) for the queried user.
2. Hash the queried user's signature bands using the same band-hash functions (total number = m / r).
3. For each band bucket, find if the queried user's band hash value is in the bucket key. If it is, add the corresponding users in the value list to a candidate list.
4. Check the Jaccard distance with the queried user for all the candidate users and return the one with smallest distance.

The time complexity will be much better than O(n) since we are just searching several buckets while it won't hurt the accuracy.

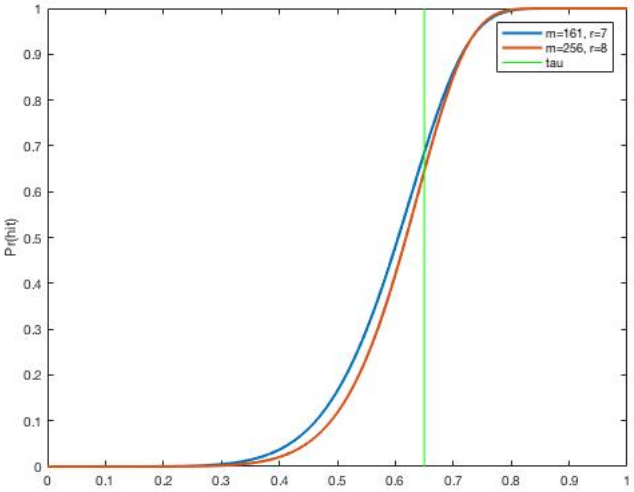


Figure 4: Comparison of m =161 and m=256