ESE 545: Data Mining **Fall 2018**

Project 1 Writeup

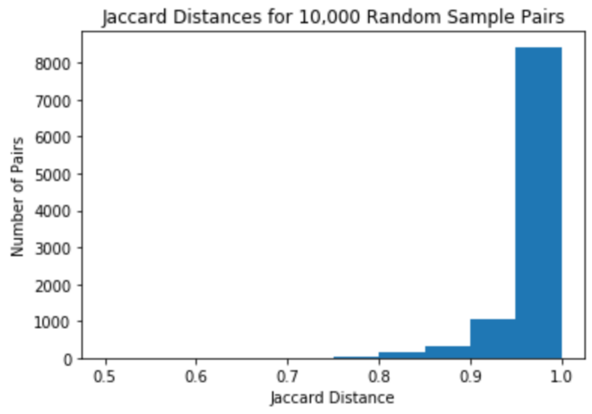
Name: *Joseph Lauigan*

1. **Problem 1**

   In problem 1 - the function that reads in the data does a single parse through the "Netflix data.txt" file. It parses each line read, then determines whether it is a movie or a user rating for a given movie. From there it tosses out the ratings that are less than 3 stars. In this initial pass through, **4,499 movies are imported, as well as 20,496,505 3-star ratings out of 24,053,764 ratings**. From here, using the Counter object derived from the Collections class, users that had more than 20 movies rated were discarded.The final implementation called the Counter object twice, one to create a user row ordering and the other to hold the numbers of movies each user has rated. This removed **241,513 users out of 467,134 - leaving the number of unique users left at 225,621**. Users were mapped to column numbers and movies were assigned row numbers (in the user index and movie column variables, respectively).

2. **Problem 2**

   In this problem, to create 10,000 unique pairs, the random function was used to sample two random user columns. Checks were put into place to ensure that there were no repeat pairings. Once that was done, the jaccard distance was computed on the raw data representation of the user columns (1/0 values). A histogram of the jaccard distances is included below. The average Jaccard distance was approximately **0.98**, and the lowest Jaccard distance was **0.5** (although this changed depending on the current run due to different pairings).

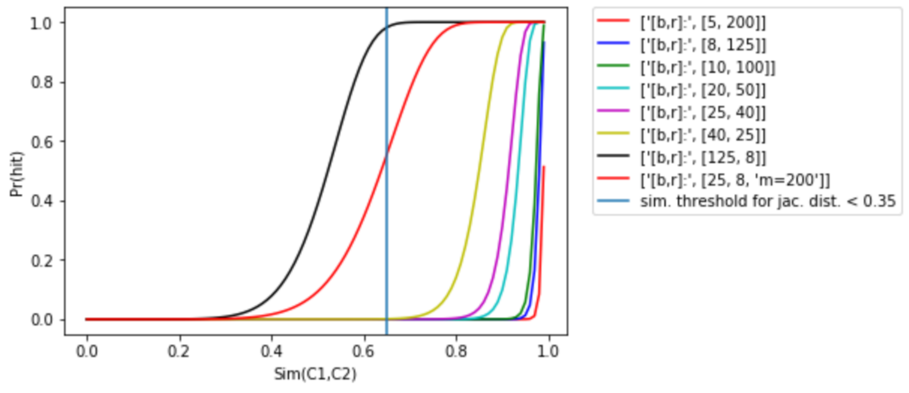


3. **Problem 3**

   A more efficient way to store the user data is to use a signature matrix, which is done by using a hashing function that does a random permutation on the ordering of the rows, and hashes the column to the value of it's first non-zero index. The signature matrix is then compiled by running m-hash functions on each user - resulting in a m-by-unique-users matrix. Since storing random permutations of matrix with a large number of rows is in-feasibly, the random permutation of each row will be modeled by $(ax+b)\%r$, where r is the number rows(and is assumed to be prime), *a,b* are random numbers between 0 and r-1, and x is the row number. The resulting value is the new row number in the permutation. For optimization, since *a,b,and r* are determined upfront, a matrix of the row mappings for each hash value is compiled into a hash values matrix. This allows for efficient computation of each user's new hash representation once the non-zero indexes are determined. List comprehension techniques were also utilized to help with runtime efficiency.

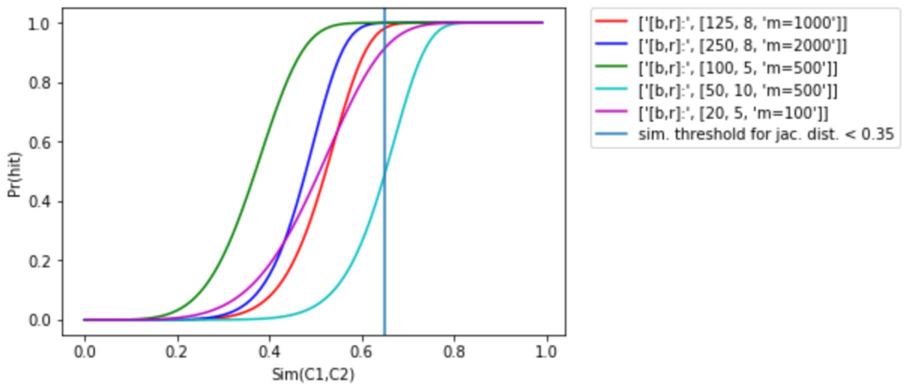
4. **Problem 4**

   Now that the signature matrix representation of the users was achieved in problem three, to find the approximate nearest neighbors of a user, locality-sensitive hashing will be implemented - which requires the signature matrix to be divided into *b-bands of r-values each.* Also, an optimal number of hash functions for the signature matrix (*denoted by m*) will need to be determined. Thus, a few graphs were compiled for different parameter values. The overall approach was to have as little false negatives as possible, thus allowing for more false positives, which will then needed to be filtered out at the end. The first graph was compiled as such, with a blue vertical line denoting the 0.65 similatiry threshold given by the problem:



   As can be seen, it looks like more bands with less values in each is a good way to go forward. Thus, a second graph was compiled to better get a sense of what parameter values would be needed. Each curve's value at sim(C1,C2) = 0.65 was printed out as well.

```
for [125, 8, 'm=1000'] value at sim(C1,C2) = 0.65 is 0.9825410061200314
for [250, 8, 'm=2000'] value at sim(C1,C2) = 0.65 is 0.9996951835326993
for [100, 5, 'm=500'] value at sim(C1,C2) = 0.65 is 0.9999955964930552
for [50, 10, 'm=500'] value at sim(C1,C2) = 0.65 is 0.4922212137958185
for [20, 5, 'm=100'] value at sim(C1,C2) = 0.65 is 0.9151289183523188
```



In the end, the parameter values of *bands = 125, r-values = 8, m = 1000* were used. The same procedure that was used in Problem 3 was implemented here to construct the signature matrix. Following this, the locality sensitive hashing technique was implemented. This required another set of random a,b pairings for each band hash function. Again, list comprehension techniques were used to improve runtime efficiency, and itertools.combinations were used to create all the different pairings for users whose bands hash to the same value. The *candidate pairs* vairable holds all the candidate pairs and the pairs are written out to a csv file.

5. **Problem 5**

The function ”nearest neighbor” accepts a list of movies a queried user has liked, and returns their approximate nearest neighbor - which is the single closet user to their movie ratings (the problem formulation asks for a single neighbor so the implementation does not return neighbors with distances less than a threshold).

Once the queried user list is passed to the function, the function does a conversion of the movie list to row indexes. Then a signature matrix column is constructed using the same a,b values as the rest of the signature matrix. This signature column is then hashed band by band. Each of these hashed band values is then used as reference for the rest of the signature matrix to see if they have the same hash value in a given band. A single pass through the signature matrix is then run, and a list of candidate users who hashed to the same band are compiled. Following this, a jaccard similiarity is then computed between the queried user's signature column and the list of candidate users. Any user who has a similarity of greater than 0.65 is then moved on to the next round, where the jaccard similarity of the binary representation of the users and queried users are then computed. The maximum of these similarities is then returned as the approximate nearest neighbor. If no users have a sim. of greater than 0.65 in the signature column representation, the user with the max of these similarities is then returned.

This function was tested by giving the function user's already in the matrix, a movie list that is identical to a user's movie list with some slight modifications, and a random movie list.