**Instruction: Completed homework should be typed (e.g., using LaTeX or word document) or hand-written clearly and scanned, and uploaded into Moodle. Other than discussion, no other form of collaboration is permitted on this assignment.**

**Q1.** Additional files required for this homework is available inside the zipped file. In this question, we are going to work through how deanonymization works on databases. We will work with a subset of the original Netflix database. You are given a set of 15 movies in the folder titled **movies**. The identifiers for these movies are [03124, 06315, 07242, 16944, 17113, 10935, 11977, 03276, 14199, 08191, 06004, 01292, 15267, 03768, 02137]. The movies folder contains csv files for each movie. Each line of the csv file has three entries: a user id, the date of rating, and the rating provided.

**Learning Objectives**: The problems in the part are based on the paper ‘Provable Deanonymization of Large Datasets with Sparse Dimensions” [1]. We will refer to the paper through all problems in this part. We will perform an attack along the lines of the original Netflix-IMDB deanonymization attack. In particular, we will learn how to identify a user by utilizing noisy and incomplete auxiliary information.

**Starter Code:** You may use the provided starter code (in Python) for this homework. The script reads each file from the movies folder and populates the database *db*. *db* is a Python dictionary. Dictionaries consist of pairs (called items) of keys and their corresponding values. Each element of *db* is the tuple *<user-id, movie-dict*>. *movie-dict* is also a dictionary representing the user’s ratings for various movies, with each item being the tuple *<movie-id, rating>.* It is not compulsory that you use the starter code provided. You can use any programming language of your choice (among C/C++/Java/Python), but you must provide a README file with the proper step by step process for running your code.

Starter code for this problem is provided in *link.py*. You are given the auxiliary information for one user (*aux*). The auxiliary information contains noisy ratings given by the user for 12 of the 15 movies. You can think of these being perturbed ratings given by a user on IMDB. This auxiliary information is provided in Table 1 and in the variable aux in *link.py*. You, as the attacker, want to identify the user id for whom the auxiliary information is provided.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Movie | Rating | Movie | Rating | Movie | Rating | Movie | Rating |
| 14199 | 4.5 | 17113 | 4.2 | 06315 | 4.0 | 01292 | 3.3 |
| 11977 | 4.2 | 15267 | 4.2 | 08191 | 3.8 | 16944 | 4.2 |
| 07242 | 3.9 | 06004 | 3.9 | 03768 | 3.5 | 03124 | 3.5 |

Table 1: Auxiliary information for the target user

(a) Using the following definition of the weigh function compute *w()* and compute the weights of each movie. Tabulate the weights obtained for each movie. This should be a table with 15 movie-ids and their corresponding weights. **[points 10]**

(b) Now complete the function *score()* and compute the similarity score of the auxiliary information with respect to every user’s ratings in the released database. Print out the top five similarity score. **[points 20]**

Where

Where *p(i)* is the maximum possible difference between values of column *i* (i.e., *max* **range** possible for a given movie even considering values from *aux).* The value of each column is scaled (normalized) by *p*, so that the value for *T (., .)* lies in the interval [0, 1].

(c)Compute the followings:

* What is the user-id of the user with the highest score with *aux*? **[points 5]**
* Write out the movie ratings of this user from the database, side-by-side with the ratings from the auxiliary data. Comment on how similar they are. **[points 5]**

(d) Compute the followings:

* What is the difference between the highest and second highest similarity score? [**points 5**]
* Assume the algorithm accepts the top candidate if the difference between the highest and second highest similarity score > γM, where is the scaled sum of weights of attributes in *aux* and γ is some constant.
  1. if γ = 0.1 would you accept the top candidate? [**points 2.5**]
  2. If γ = 0.05 would you accept the top candidate? [**points 2.5**]

**Q2.** For the following questions consider table below-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | ZIP code | Age | Salary | Disease |
| 1 | 47677 | 29 | 3K | Gastric ulcer |
| 2 | 47602 | 22 | 4K | gastritis |
| 3 | 47678 | 27 | 5K | Stomach cancer |
| 4 | 47905 | 43 | 6K | gastritis |
| 5 | 47605 | 30 | 7K | flu |
| 6 | 47906 | 47 | 8K | bronchitis |
| 7 | 47674 | 36 | 9K | bronchitis |
| 8 | 47607 | 32 | 10K | pneumonia |
| 9 | 47909 | 52 | 11K | Stomach cancer |

(a) What are the quasi-identifiers and sensitive attributes in this table? **[points 5]**

(b) Your task is to compose a 3-anonymous, 3-diverse table.

* Define the generalization hierarchy for each of your quasi-identifiers. **[points 5]**
* Next, using incognito algorithm draw the generalization lattice (remember to label or color each node as to whether they satisfy 3-anonymity or not). **[points 10]**
* Identify the node in the lattice that you are using for your final solution and write out the final table. **[points 10]**

(c) Compute the followings:

* Compute the t-closeness of your solution (from part **b**) with respect to **only** ‘salary’. **[points 10]**
* Does your solution resolve the ‘similarity attack’ with respect to the sensitive attribute ‘disease’? If not, compute an alternative solution (reusing the generalization lattice in part **b**). If yes, please state why is resolves **‘**similarity attack’. **[points 10]**

**Submission:**

You have to submit **three** files:

1. Merge all the written parts into a single pdf file named <your unity id>\_HW1.pdf.

2. Rename the program file you used for as <your unity id>\_HW1.extension (e.g., .c/.cpp/.java/.py).

3. Add a README file regarding how to run your code.

Zip all files into <your unity id>\_HW1.zip and submit the zip file on Moodle.