COMS 6998-2 Social Networks 14Fall

Challenge 2. Identity Reconciliation

Due: Wednesday Dec. 3rd 11:55 pm EST

Motivation & Background

In this challenge, we will delve into the consequence of using multiple service on your overall anonymity and identity. Let us take an example. Alice is a social media user and she maintains online public activities on sites like yelp (where she reviewed restaurants she likes or not), Facebook (where she posts pictures of family and friends gathering), Instagram (where she selects a set of pictures that more artistically oriented to her set of followers). She also has social media she uses for her professional or social project (she is a well known environmental conscious blogger who posts about her everyday experience), she may also have another blog with geographical activities (since she is a classical music critics, she usually posts stories from concert in leading capitals and music festival). *A priori*, when it comes to identity, there could be two situations: Alice may sign all her contributions by the same unique name and linked all her accounts. But for various reasons, she may decide to distinguish her identities as, perhaps her followers of music critics might not be too excited by her photographic talents, or they may disagree with her gastronomical taste. But can Alice really do that successfully? Isn't possible, based on the information she publicly release under two identities, to find that they share a common unique patterns (*e.g.*, a visit to the Provence region during a festival followed by a trip to Tokyo to meet with a leading director, before an ecology festival organized in New Zealand).

From the point of view of a social media provider, this problem can be posed as follows. Assuming you have two populations of users leaving activities in two different domains. Assume for simplicity that the populations are identical (because everybody uses this service extensively, or because you have a way to select users appropriately for that). Is it then possible to reconcile the identity of each users, and pair them without making a single mistake to be able, for instance, to characterize them from multiple angles. This, which affects the

ability of Alice to truly hide herself, is called the identity reconciliation problem. It is a fundamental problem, it has been studied in multiple contexts, and so far, most of the methods trying to solve it across multiple domains have been shown to be very limited.

This challenge contains three tasks. We encourage you to read the reference papers that will also be discussed in class, to think about the algorithms they propose and their limitations. If you find yourself thinking "this paper clearly is not making the best choice as one could do X instead", great news! You have a chance to do exactly that and prove your point by answering the three tasks we develop.

Task 1. Two social networks

Related Papers

Korula, Nitish, and Silvio Lattanzi. "An efficient reconciliation algorithm for social networks." arXiv preprint arXiv:1307.1690 (2013).,

Yartseva, Lyudmila, and Matthias Grossglauser. "On the performance of percolation graph matching." Proceedings of the first ACM conference on Online social networks. ACM, 2013

In this task, you're given two social graphs G1 and G2, both are generated from a 'real' social graph G, in the way described in section 3.1 of Korula (the independent edge deletion model), with probability s1=s2=0.7. G is an Erdős–Rényi random graph G(50000, 0.0004). You're also given the set of initial seed of links, L, with linking probability l=0.005. (Read *Yartseva* 6.1 to see why we use those numbers)

Your task is, output all (u, v) pairs that you think u and v represent same person in G, where u ∈ G1 and v∈ G2.

G1 and G2 are stored in G1.txt and G2.txt respectively. L is stored in L.txt. You can assume these files are in the working directory when your program is running. Your output should be written to output.txt, in the same format as L.txt.

Example G1.txt

1 3

1 2

3 4

Example G2.txt

1 2

4 2

2 3

3 4

Example L.txt (each line contains “a b”, which means a in G1 is b in G2)

2 3

Example output.txt (order of lines are not important)

2 3

4 1

1 4

3 2

(The example here has much smaller G than the actual test data)

Task 2. Only a small piece of your data was shared

Related Paper

Rossi, Luca, and Mirco Musolesi. "It's the way you check-in: identifying users in location-based social networks." Proceedings of the second edition of the ACM conference on Online social networks. ACM, 2014.

In this task, you’re given two sets of check-in data N1 and N2. N1 and N2 are generated from an original check-in data set N in the following way.

For each user in N, we give him/her a randomly assigned new user id, and after that we pick out n of his/her check-in records. We set the user id of those

records to be the new user id ('anonymization'), and then put them into N1. The remaining records form N2. Users who have less or equal to n check-in records are ignored.

Your task is, given N1 and N2, output all (u, v) pairs that you think u and v represent same person, where u ∈ $user\ ids\ of$ N1 and v ∈ $user\ ids\ of\ N2$.

N1 and N2 are stored in N1.txt and N2.txt respectively. You can assume N1.txt and N2.txt are in the working directory when your program is running. Your output should be written to output.txt, in the same format as output of Task 1.

Example N1.txt (each line contains 'user_id time latitude longitude location_name', separated by \t)
43502 2009-09-27T07:23:21Z 33.597754 130.332141 fff
43502 2009-09-28T08:01:11Z 33.597754 130.332141 fff
43503 2009-09-27T05:59:21Z 33.595195 130.322746 bbb
43503 2009-09-28T11:17:36Z 33.594491 130.397268 aaa

Example N2.txt
12345 2009-09-26T06:47:14Z 33.593535 130.322841 ccc
12345 2009-09-26T10:58:53Z 33.594491 130.397268 aaa
12345 2009-09-27T22:31:39Z 33.594491 130.397268 aaa
12345 2009-09-26T06:51:21Z 33.595195 130.322746 bbb
12346 2009-09-25T09:00:06Z 33.595231 130.330412 ddd
12346 2009-09-25T12:33:32Z 33.598141 130.334123 eee
12346 2009-09-25T10:29:45Z 33.595195 130.322746 bbb
12346 2009-09-25T07:28:53Z 33.597754 130.332141 fff

Example output.txt
43502 12346
43503 12345

Task 3. A year's data was stolen

In this task, you're given two sets of check-in data, both generated from an original data set N in the following way.

For each user in N, we split that user's record into two parts. The first part contains check-in recorded in 2009, and the second part contains check-in recorded in 2010. The first and the second parts must have at least 5 records otherwise that user is ignored. The first set of check-in data, C1, contains all users' first parts of records, and the second set, C2, contains all second parts. Like what we have in Task 2, the user ids in C1 are 'anonymized' (injectively mapped to random numbers).

Again, your task is output all (u, v) pairs that you think u and v represent same person, where u ∈ $user\ ids\ of$ C1 and v ∈ $user\ ids\ of\ C2$.

C1 and C2 are stored in C1.txt and C2.txt respectively. You can assume C1.txt and C2.txt are in the working directory when your program is running. Your output should be written to output.txt, in same format as L.txt (of task 1).

Example C1.txt

48121 2009-09-26T06:47:14Z 33.593535 130.322841 ccc
48121 2009-09-26T06:51:21Z 33.595195 130.322746 bbb
48121 2009-09-26T10:58:53Z 33.594491 130.397268 aaa
51231 2009-09-25T10:29:45Z 33.595195 130.322746 bbb
51231 2009-09-25T07:28:53Z 33.597754 130.332141 fff

Example C2.txt

12345 2010-09-27T05:59:21Z 33.595195 130.322746 bbb
12345 2010-09-27T22:31:39Z 33.594491 130.397268 aaa
12345 2010-09-28T11:17:36Z 33.594491 130.397268 aaa
12346 2010-09-25T09:00:06Z 33.595231 130.330412 ddd
12346 2010-09-25T12:33:32Z 33.598141 130.334123 eee

Example output.txt

48121 12345

51231 12346

Grading

Note: It's important that you follow all input and output formats strictly, as we will use scripts to test your program and check your output.

Task 1 (30%)
We will generate 20 Erdős–Rényi random graphs, and their G1 and G2, to test your program.
Task 2 (30%)
We will generate 40 N1 and N2 pairs from the attached Brightkite dataset, with n, the number of records being picked out for each user, ranges from 1~20.
Task 3 (30%)
We will generate C1 and C2 from the attached Brightkite dataset.
Documentation & proper input/output format (10%)
Do not forget to write your README file nicely, as well as follow all I/O formats.

For each task, your scores will be based on your average F1 score (harmonic mean of precision and recall), and they will be rescaled so that all submission that work reasonably well for the simplest case will have at least 50%, and the best ones will get 100%.

Please note that, for Task 2 and 3, although we give you the complete Brightkite dataset, you're not allowed to access it in your algorithm. Hardcoding any part of it into you code is banned too. We will check your code to ensure this. (Don't spoil all the fun!)

You're encouraged to generate test cases by yourself and use them to see how good is your algorithm, and probably with some simple improvements, you will get better results than the papers.

Deliverables

Please submit all of your code and related files as a zip file with its file structure

looks like this:

abc123_challenge2.zip (uni_challenge2.zip)

| README

| task1 (directory)

| | main.c

| | Makefile

| | ...

| task2 (directory)

| | main.c

| | Makefile

| | ...

| task3 (directory)

| | main.c

| | Makefile

| | ...

Again, you can use any programming language you like. You code will be tested on CLIC machines and it's your responsibility to make sure that your code compiles and runs there. For platform spec check HW1 handout part B.

Include a README file to show how to compile and run your program for each task, and explain your algorithms and analyses in brief. If you find your algorithm works better than the original papers', include performance comparison in the README too. If you have generated test cases and tested your program by yourself please also include all related code in the zip and write your test results in the README file.