Project 1: Inference in Location-Based Social Networks

This project is due on **January 30, 2018** at **11:59pm CST**. We strongly recommend that you get started early. You will get 80% of your total points if you turn in up to 72 hours after the due date. Late work will not be accepted after 72 hours past the due date.

The project is split into two parts. Checkpoint 1 helps you to get familiar with the language and tools you will be using for this project. The recommended deadline for checkpoint 1 is **January 16, 2018**. We strongly recommend that you finish the first checkpoint before the recommended deadline. However, you do NOT need to make a separate submission for checkpoint 1. That is, you need to submit your answers for both checkpoints in a single folder before the project due date on **January 30, 2018** at **11:59pm CST**. Detailed submission requirements are listed at the end of the document.

This is an individual project; you SHOULD work **individually**.

The code and other answers you submit must be entirely your own work, and you are bound by the Student Code. You MAY consult with other students about the conceptualization of the project and the meaning of the questions, but you MUST NOT look at any part of someone else's solution or collaborate with anyone else. You may consult published references, provided that you appropriately cite them (e.g., with program comments), as you would in an academic paper.

Solutions MUST be submitted electronically in your svn directory, following the submission checklist given at the end of each checkpoint. Details on the filename and submission guideline is listed at the end of the document.

Release date: January 16, 2018
Checkpoint 1 Recommended Due date: January 16, 2018
Checkpoint 2 Due date: January 30, 2018 at 11:59pm CST

Course SVN: To be announced

Introduction

In this machine problem you will explore inference of private attributes in social networks. You are given real anonymized datasets of two location-based social networks. The datasets contain friendship information (i.e., which users are friends), as well as the home location (i.e., GPS coordinates) of a subset of the users. Some but not all the users have shared the location of their homes on the social network. Your task is to leverage the available information (i.e., home locations and friendships) to infer the home locations of users who have chosen not to share it.

This assignment is individual. Your implementation must use Java.

Please read the assignment carefully before starting the implementing.

Objectives

- Understand homophily in social networks.
- Be able to infer private attributes from real social network datasets.
- Gain familiarity with Java programming.

Checkpoints

This machine problem is split into 2 checkpoints. Checkpoint 1 would help you get familiar with Java programming, the social network datasets, and the provided code. You will need to work on the provided code to answer a few questions and print the results to a file. In Checkpoint 2, you will implement 2 inference algorithms. In Section 1.2.1, you will implement a simple inference algorithm and evaluate it using the code provided. In Section 1.2.2, your goal is to improve on the accuracy of the algorithm of part 1. For this you will be free to implement the inference algorithm of your choice.

Datasets

Dataset 1 contains the (correct) home locations of all users, but a subset of users are specified as having chosen not to share their home locations. For those users home locations are meant to be used as "ground truth" for evaluation. Dataset 2 has the same format as dataset 1 but it does not contain the home locations of those users who have decided not to share it.
Each dataset contains two files: friends.txt and homes.txt. friends.txt contains the friendship information, i.e., the edges of the social graph in the format: '<userId1>, <userId2>', one edge per line. Each user has a unique userId which is a non-negative integer.
homes.txt contains the home locations of users. Each line has the format: '<userId>, <latitude>, <longitude>, <homeShared>', if the home location information is available; or '<userId>', if the home location is not available. The home location is given as a latitude-longitude pair both of which are represented as floating-point numbers. In case the home location is available, homeShared is 0-1-valued and indicates whether the user has shared his/her home location.

When the home location of a user is available in the dataset, but not shared (i.e., homeShared is '0'), the idea is to "pretend" that it is not available to the inference algorithm (but use it only as part of the ground truth).

Code Skeleton

To help you with the implementation, you are provided with a code skeleton which prototypes the main classes, implements the inference evaluation logic, and includes some useful methods.
You are free to modify the provided code as long as the interface-like functionality of the `MP1` class is preserved and fully implemented. Keep in mind that part of the assessment of this assignment will test the functionality of objects returned through the `MP1` class. Therefore you should implement the functionality of all objects directly or indirectly accessible from the methods of `MP1.java` according to their specification. (In particular, if you decide to subclass some of them, make sure to respect the Liskov substitution principle.)

1.1 Checkpoint 1 (10 points)

The first checkpoint is designed to help you get familiar with Java programming and the social network datasets. We have provided a code skeleton in Java for both checkpoints and two scripts (compile1.sh and run1.sh) for checkpoint 1. Your code must compile and run on the Linux EWS machines without root permission. If you need special compilation or run directives, modify the scripts compile1.sh and run1.sh accordingly.

1.1.1 Java Tutorial (0 points)

This section is designed to help you get familiar with Java. If you are proficient in Java, you can skip to Section 1.1.2. No submission is required for this section.

1.1.1.1 Java Environment

Check your Java environment with command `java -version` and `javac -version`. You can use any version of Java, but your code must compile and run on the Linux EWS machines with Java 8. If you do not know how to install JDK and set up the environment, check the following reference.

Reference

- JRE and JDK Installation. `http://docs.oracle.com/javase/8/docs/technotes/guides/install/install_overview.html`

1.1.1.2 File Input and Output

In this machine problem, you will need to read and parse the datasets from a file. You can use the `Java.io.BufferedReader` class to achieve this. The `BufferedReader` class reads text from a character-input stream. In our case, the input stream would be a `Java.io.FileReader` class reading from the provided file paths (e.g., `./dataset1/homes.txt`).

References

- `BufferedReader` API.

  `https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html`

- `FileReader` API.

  `https://docs.oracle.com/javase/8/docs/api/java/io/FileReader.html`

1.1.1.3 HashMap

In this project, you need to use `Java.util.HashMap<K,V>` to store the `<UserId, Location>` pairs in `GroundTruth` class and the `<UserId, User>` pairs in `SocialNetwork` class. A `HashMap` is the Hash table based implementation of the `Map` interface. It maintains key-value pairs and usually has constant time complexity for storage and retrieval.
Note that the keys and values in a `HashMap` both need to be Java objects. That is, you cannot add primitive types (e.g., `long`, `int`) into a `HashMap`. Instead, you should use their object wrappers (e.g., `Long`, `Integer`). Java supports automatic conversion between a primitive type and its object wrapper (i.e. autoboxing and unboxing).

References

- `HashMap` API. `https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html`
- Primitive Wrapper Classes. `https://en.wikipedia.org/wiki/Primitive_wrapper_class`
- Autoboxing and Unboxing. `https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html`

What to submit No submission required.

1.1.2 Practice on the Social Network Dataset (10 points)

In this section, you need to complete the implementation of `DatasetReader`, `GroundTruth`, `SocialNetwork`, `User`, and `DatasetAnalyzer` in the provided code skeleton. You need to submit the output file of your code.

1.1.2.1 Read and Parse the Dataset

Implement the `read()` method of the `DatasetReader` class. Your code should work for both datasets.

Tips:

- Locations read from `homes.txt` should be parsed into a `GroundTruth` object with at least one attribute `homes` (`Map<Long, GPSPoint>`).
- Data read from `friends.txt` and `homes.txt` should be parsed into a `SocialNetwork` object with at least one attribute `nodes` (`Map<Long, User>`).
- Each user in the dataset should be stored as a `User` object with at least three attributes: `id` (`long`), `home` (`GPSPoint`), and `friends` (`Set<User>`). These attributes should come from both `homes.txt` and `friends.txt` files.
- For users whose home locations are not shared, their `home` attributes should be set to `null`.

1.1.2.2 Implement Main Data Structures

- Implement the `getHomeLocation()` and `setHomeLocation()` methods in the `GroundTruth` class.
- Implement the `User` class.
- Implement the `SocialNetwork` class. Make sure that `getNodeById()` runs in constant time, and that `getFriends()` runs in time at most linear in the number of friends.

1.1.2.3 Implement the `DatasetAnalyzer` Class

Answer the following questions by implementing the methods in the `DatasetAnalyzer` class. The `printResult()` method will print your answers to a file called `checkpoint1_results.txt`.

Questions:

1. How many users are there in dataset 1?
2. How many users in dataset 1 have unknown locations?
3. How many users in dataset 1 have unknown locations and no friends?
4. What is the baseline accuracy (p_b) (accuracy of trivially predicting all unknown locations as `(0,0)`) for dataset 1? You can assume that no user in dataset 1 has home location (0,0).
5. What is the upper bound on inference accuracy (p_{u_1}) for dataset 1 if we assume that for a given user, we can correctly infer location if unknown UNLESS that user has NO friends.

6. What is the upper bound on inference accuracy (p_{u_2}) for dataset 1 if we assume that for a given user, we can correctly infer location if unknown UNLESS that user has NO friends who shared their locations.

Tips:

1. In the calculation of inference accuracy, users who have shared their home locations are also included. That is,

$$\text{Accuracy} = \frac{\text{number of shared home locations} + \text{number of correctly inferred unknown locations}}{\text{number of users in the dataset}} \tag{1.1}$$

2. You DO NOT need to implement any inference algorithms to answer these questions.

3. For more information about how the inference accuracy is calculated, please refer to the `EvaluateInference()` method in the `Evaluator` class.

1.1.2.4 Compile and Run

Compile and run your code with `compile1.sh` and `run1.sh`. The answers for the above questions will be written into a file called `checkpoint1_results.txt`. You need to submit the output file and your code for checkpoint 1.

What to submit

- Create a directory called `mp1` under your subversion repository.
- Place `checkpoint1_results.txt` in `mp1`.

1.2 Checkpoint 2 (90 points)

You DO NOT need to create a separate Java project for Checkpoint 2. You NEED to continue on your implementation in Checkpoint 1.

Again, We have provided a code skeleton in Java for both checkpoints and two scripts (compile2.sh and run2.sh) for checkpoint 2. Your code must compile and run on the Linux EWS machines without root permission. If you need special compilation or run directives, modify the scripts compile.sh and run.sh accordingly. For this checkpoint, you will implement two inference algorithms to infer the locations of users in dataset 1 and dataset 2.

1.2.1 Simple Inference (20 points)

In this first part you will implement the simple inference of Algorithm 1.

Algorithm 1 Infer home location of user u

```
if homeLoc(u) is shared then
    return homeLoc(u)          {Home location is shared, no need to infer it.}
end if
f ← friendsWithSharedHomeLoc(u)
h ← geographicCenterOfHomes(f)
return h
```

Note that the geographic center can be defined in several ways. To simplify the implementation, you may compute it as the centroid on a flat rectangular projection.

1.2.1.1 Implement the Inference Algorithm

Implement the `inferHomeLocation()` method of the `SimpleInferenceAlgorithm` class according to Algorithm 1.

1.2.1.2 Test Your Implementation

You may use the tests provided in `test/MP1Tests.java`. In particular, the `Within25kmAccuracy` test is the primary way to evaluate your algorithm. It computes the inference accuracy as the proportion of users correctly inferred to reside within 25km of their true home.

1.2.2 Improved Inference Algorithm (30 points)

In this part, you are free to implement the inference algorithm of your choice. The goal is to improve the inference accuracy.
To get you started, here are some suggestions of possible improvements to the algorithm of part 1.

- **Leverage shared home locations of friends of friends:** For example, if the number of friends having shared their home locations is below a certain threshold, your algorithm could compute the geographic center of the shared home locations of friends of friends instead.
- **Leverage proximity between shared home locations of friends:** If you think that typical users have many friends in their home city, and only a few friends in distant places, then your algorithm's inference strategy could be to take the geographic center of the smallest clusters of friends' home locations.
- **Leverage dependence between inferences:** The algorithm of part 1 makes independent inferences for each user. Your algorithm could use past inferred home locations as a way to increase the number of shared friends locations.

You may also search the research literature for existing techniques.

1.2.2.1 (Optional) Use the Visualization Class

In order to get started and improve on the algorithm of the first part, you may use the `Visualizer` class provided. The `drawFriendsMap()` and `drawInferenceMap()` methods allow you to visualize the home locations of users (and of their friends) on a map. These methods use Google Maps JavaScript v3 API to create a HTML file that can be opened with a Web browser.

1.2.2.2 Subclass and Implement Your Inference Algorithm

Create a class that extends `InferenceAlgorithm` and implements its `inferHomeLocation()` method. Make sure to complete the implementation of the `MP1` class accordingly. You may use the `distanceTo()` method of the `GPSPoint` class which calculates the distance (in km) between two latitude-longitude pairs.

1.2.2.3 (Optional) Evaluate Your Algorithm Using Dataset 1

Since the second dataset does not have ground truth, you will not be able to evaluate directly your algorithm. However, you may evaluate it on the first dataset, for example with the `Within25kmAccuracy` test.

Tips:

- Part of your implementation will be tested using automated tests, some of which are provided in the code skeleton.
- We will not evaluate the robustness of your implementation by using improper inputs (e.g., incorrect dataset format). Nevertheless, you should make sure that your code handles errors gracefully in such cases and does not fail silently.

1.2.2.4 Ranking

In addition to grading your assignment we will rank your part 2 algorithms using the ground truth of dataset 2 according to various metrics (e.g., the `Within25kmAccuracy` test). A few of the highest ranking entries will be presented to the class.

What to Submit

- Create a `code` folder under `mp1` in your subversion repository.
- Put your implementations under the `code` folder.
- `code` should contain your implementations and scripts for BOTH checkpoints.

1.2.3 Report (40 points)

As part of this assignment you must write a one-page report in 11pt font (not including reference). The report must explicitly answer questions in Section 1.2.3.1, Section 1.2.3.2, and Section 1.2.3.3 below. You may include additional information relevant to the evaluation of your assignment, provided the report does not exceed one page in length.

1.2.3.1 Question 1: Dataset Analysis (10 points)

What is your answer for p_b, p_{u_1}, and p_{u_2} in Section 1.1.2.3? Do you think it is possible to get an inference accuracy higher than p_{u_1} and p_{u_2}? Why or why not?

1.2.3.2 Question 2: Simple Inference (10 points)

What is the accuracy obtained through the `Within25kmAccuracy` test with your implementation of part 1? How good is Algorithm 1? (Compare the obtained accuracy with the baseline accuracy you calculated in Section 1.2.3.1)

1.2.3.3 Question 3: Improved Inference Algorithm (20 points)

Describe briefly the algorithm you implemented in part 2. Why do you think it will yield better inference accuracy than Algorithm 1? In which cases do you think your algorithm will perform well/poorly?

What to Submit Place your report in the `mp1` directory, name the file `report_[your_netid].pdf`, and put your name and netid at the top of the document.

Submission Checklist

Place the following files in `mp1` in your subversion repository.

- `report_[your_netid].pdf` (40 points) [Report for Section 1.2.3]
- `checkpoint1_results.txt` (10 points) [Answers for Section 1.1.2.3]
- `code` (50 points) [Codes for Section 1.1, Section 1.2.1, and Section 1.2.2]
  - `src`
  - `libs`
  - `dataset1`
  - `dataset2`
  - `compile1.sh`
  - `run1.sh`
  - `compile2.sh`
  - `run2.sh`