Project 1: Inference in Location-Based Social Networks

This project is due on **February 17, 2023** 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. Please let the TAs know if you decide to take this option (email or private campuswire post). 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 30, 2023**. 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 **February 17, 2023** 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 git 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 23, 2023 at 10:00am CST
Checkpoint 1 Recommended Due date: January 30, 2023
Checkpoint 2 Due date: February 17, 2023 at 11:59pm CST

Course repo setup: `https://edu.cs.illinois.edu/create-gh-repo/sp23_cs463`
Course assignment repo: `https://github.com/illinois-cs-coursework/sp23_cs463_.release`
Your submission repo: `https://github.com/illinois-cs-coursework/sp23_cs463_[NetID]`

Introduction

In this machine problem, you will explore the 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 users have shared the location of their homes on 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 be compatible with the provided Python3 code skeleton and interfaces it defined. ***Please read the assignment carefully before starting the implementation.***

Objectives

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

Checkpoints

This machine problem is split into 2 checkpoints.

In Checkpoint 1, you will get familiar with Python programming, the social network datasets, and the provided code skeleton and interfaces. You will need to implement the parser of given datasets based on the given code skeleton, and answer a few questions about the datasets.

In Checkpoint 2, you will implement 2 inference algorithms. In Section 2.1, you will implement a naive inference algorithm which is defined by the given pseudo code in Section 2.1. In Section 2.2, you are to implement an improved inference algorithm of your choice that should outperform the given naive inference algorithm.

Datasets

As previously described, there are 2 datasets provided. Dataset 1 provides true home locations of all users, but a subset of users choose to NOT have their home locations shared publicly. 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 consists of 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 that prototypes the main classes, implements the inference evaluation logic, and includes some useful methods.

In the distributed folder, you should find the following folders and files:

- cp1.py, cp2.py

  All your graded work should be in these 2 files.

  These are the ONLY files considered as your submission.

- main.py

  This is the provided executable script that calls your implementations in cp1.py, cp2.py. You should be able to use it for debugging. You are free to edit this file as you wish, and this file will **NOT** exist in the grading environment.

  Usage: `./main.py <cp1|cp2> /path/to/homes.txt /path/to/friends.txt`

  - With the first argument being `cp1`, it calls your parsers for homes.txt and friends.txt in sequence, then call your `cp1_3_answers()` function and print out the results it returns.
  - With the first argument being `cp2`, it calls the two inference algorithms you implemented in cp2.py, and then print results returned by your accuracy calculation function `cp2_calc_accuracy()` in cp2.py.

- user.py, utils.py, plot.py

  user.py defines the data structure depended on the given code skeleton.

  utils.py provides you a helper function for calculation distances between 2 GPS points.

  plot.py provides you a helper function for plotting user and their friends on a world map.

- dataset{1,2}/*

  Where the provided homes.txt and friends.txt are located for each dataset.

- requirements.txt

  This provides the libraries that can support most functions you may use in this MP and will be installed in the grading environment. **Please do not import libraries that are not on this list** because the grading environment may not have such libraries which can make your code crash, resulting in no grade.

Checkpoint 1 (10 points)

Checkpoint 1 is designed to help you get familiar with Python programming and the social network datasets. We have provided a code skeleton in Python for both checkpoints. Your code must be able to run on the Linux EWS machines. Unless otherwise permitted by the course staff in charge, your code should not depend on other packages or libraries.

Data Structures

In checkpoint 1, you need to complete the parsers for the dataset and store all the users parsed in a dictionary of `User` objects. For more information about Python's dictionary data structure, you may click on `dict()`. It applies to other Python data structures mentioned further in this document as well.

The Python dictionary stores objects in key-value pairs. In this case, your key should be each user's `id`, and the value stored should be the `User` object defined by user.py. The `User` object consists of the following properties:

- `id` – id of the user.
- `home_lat, home_lon` – latitude and longitude of the user's home location.
- `friends` – a `set()` containing each user id of the user's friends.
- `home_shared` – if the home location of this user is shared publicly.

In addition to these given properties, the User object also provides two helper functions, constructor `__init()__`, and `latlon_valid()` which indicates whether valid coordinates are stored in the `home_lat, home_lon` properties of this object.

1.1 Read and Parse the homes.txt (2 points)

Grading Rubric: Each dataset is worth 1 pt. Any misalignment for `id,home_lat,home_lon and home_shared` will result in a 0 pt for that dataset.

Your first task in this machine problem is to implement the parser for homes.txt. You should

implement the parser within the given function `cp1_1_parse_homes()`. The provided code has already created an empty Python dictionary object for you in variable `dictUsers_out`. Following the initiation of `dictUsers_out`, you are also provided a `for` loop which iterates through the whole homes.txt, one line per iteration. The iterator `i` stores a `Lists` object, and each of its elements represents a cell from the csv line, in order.
Your task is to initiate a `User` object with `id, home_lat, home_lon, home_shared` from the file input, store it into `dictUsers_out`, and return `dictUsers_out` in the end. Please follow the constructor of the User class when you parse the dataset file. That being said, you should convert `user_ID` to integer, `location` to floating values, and `home_shared` to bool or integer (0 or 1).

Tip: Your implementation should work for *both* dataset1 and dataset2. In dataset 2, users without a shared home location may not have the same number of cells as those with a home location shared.

1.2 Read and Parse the friends.txt (2 points)

Grading Rubric: Each dataset is worth 1 pt. Any misalignment for `friends` will result in a 0 pt for that dataset.

You should implement the parser for friends.txt within the given function `cp1_2_parse_friends()`. The function takes two arguments `f, dictUsers`. The `f` is path to friends.txt which has already be opened by the provided code just as in the previous task. The `dictUsers`, is a dictionary that stores `User` objects you've just created with `cp1_1_parse_homes()`.
As each line of friends.txt represents a pair of friends, your task here is to update `User.friends` property for each of the two friends.

Tips:

- You may assume all users referenced in a friends.txt exist in homes.txt.
- `friends` is a `set()` of user ids in int type, so you don't have the need of dealing with duplications nor copying and/or referencing User objects.

1.3 Answer Questions (6 points)

Grading Rubric: Each question is worth 1 pt. The answer to each question is fixed since the dataset is explicit.

Return your answers to the following questions *in the given order* using function `cp1_3_answers()`. You may either calculate answers based on the `dictUsers` passed in, or hard code the values. However, your values **MUST** in numeric types, **NOT** strings. For accuracy values, we expect a floating point value within [0,1] with less than 0.01 difference to the exact value.

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 incorrectly predicting all unknown locations) for dataset 1?
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:

- Users shared their home locations need to be included when calculating inference accuracy:
$$\text{Accuracy} = \frac{\text{number of shared home locations} + \text{number of correctly inferred unknown locations}}{\text{number of users in the dataset}} \tag{1.1}$$
- You DO NOT need to implement any inference algorithms to answer these questions.

Checkpoint 2 (90 points)

For this checkpoint, you will implement two inference algorithms to infer the locations of users in dataset 1 and dataset 2.
Again, We have provided a code skeleton in Python for checkpoint 2. Your code must be able to run on the Linux EWS machines. Unless otherwise permitted by the course staff in charge, your code should not depend on other packages or libraries.

2.1 Simple Inference (35 points)

Grading Rubric: For the simple inference algorithm, the accuracy should be a fixed number because both the algorithm and dataset are fixed. Any other answer of accuracy will result in 0 pt.

In this first part, you will implement the simple inference algorithm shown as Algorithm 1.
You should implement this algorithm function `cp2_1_simple_inference()`. Since after the inference algorithm coordinates stored in `User` objects may be updated, you should duplicate the existing `User` objects and store the inferred results in a fresh dictionary, instead of updating the existing ones. The returned dictionary should include **ALL** users from the `dictUsers` passed in, not only those with inferred home locations. As the geographic center can be defined in several ways, you may compute it as the centroid on a flat rectangular projection for simplicity.

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
$f \leftarrow$ friendsWithSharedHomeLoc(u)
$h \leftarrow$ geographicCenterOfHomes(f)
return h

2.1.1 Test Your Implementation

In order to test your implementation, you may use the provided accuracy function `cp2_calc_accuracy()`. In the grading environment, we will use the same accuracy function. The accuracy is rated by measuring the inference accuracy as the proportion of users correctly inferred to reside less than 25km from their true home.

2.2 Improved Inference Algorithm (45 points)

Grading Rubric: For the improved inference algorithm, the accuracy should be better than the simple inference algorithm.

- Any accuracy below or equal to the baseline will result in a 0 pt.
- Any accuracy between baseline (not included) and 0.65 will be given 90% pts.
- Any accuracy between 0.65 (not included) and 0.68 will be given 93% pts.
- Any accuracy between 0.68 (not included) and 0.7 (not included) will be given 97% pts.
- Any accuracy larger than or equal to 0.7 will get full 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 for possible improvements to the algorithm of part 1.

- **Leverage shared home locations of friends of friends:** For example, if the number of friends had 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.

(Optional) Use the Visualization Function to Optimize

In order to get started and improve on the algorithm of the first part, you may use the provided `draw_map()` function in `utils.py`. The function takes two arguments, `uid, dictUsers`. The `uid` is the user id that you would like to plot against, and the `dictUsers` is the dictionary that contains all the user objects. The `draw_map()` function will automatically plot the uid in red and its friends in yellow on a world map. Feel free to modify the function to adapt to your needs and use it for debugging. However, this function will not be available in the grading environment, nor will your change be preserved.
NOTE: to use the `draw_map()` function, you need to install two dependency libraries: `matplotlib` and `basemap`. You may follow the steps to install these libraries:

1. Install Anaconda here: `https://www.anaconda.com/products/distribution`.
2. Create a virtual environment named as cs463 with Python 3.8.8: `conda create -n cs463 python==3.8.8`
3. Activate the virtual environment: `conda activate cs463`
4. `pip install matplotlib`
5. `conda install basemap`

(Optional) Evaluate Your Algorithm Using Dataset 1

Since the second dataset does not have ground truth, you will not be able to evaluate your algorithm directly. However, you may evaluate it on dataset 1.

Tip: 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.

2.3 Answer Questions (10 points)

For this part, please go to Coursera Week 2: “MP1: Home Location Interference” to answer all the questions. **Please note that you can make a one-time submission only**.

Submission Instructions

Place the following files in the `mp1` folder of your git repository.

- `cp1.py`
- `cp2.py`

Then use the following commands to push your submission for grading (do this often for better version control). The grading will be based on the latest submission before the deadline.

```
git add -A
git commit -m "REPLACE THIS WITH YOUR COMMIT MESSAGE"
git push origin main
```