POI Checking-In Data Report

TomatoEater 2021/12

This is an app for data analysis and visualization of POI checking-in data, which is also the final project of SJTU CS2309@Fall 2021 (Principles and Practice of Problem Solving). Here is a lab report. If you are wondering how to use the program, refer to the `README.md`.

1. Introduction

There are total nine functions in my program, which can be divided into five categories.

1. **Filter**

   Import the selected fields of user's interest according to **any** combination of *User_ID*, *POI_ID*, *GPS Range* and *Time*.

2. **Spatio-temporal Information**
   - **User's Favorite**: A table is used to display the top 5 POIs visited by specific users every month.
   - **Checking-in Number**: A line chart is used to display the number of checking-ins of specific POIs every month or every hour in a day.
3. **Comparison**
   - **DAU (Daily Active Users)**: Two line charts are used to compare the two different POIs' everyday visiting number in one month.
   - **Users' Difference**: A map and markers are used to show where two users have been in a specific month.
4. **Map-related Analysis**
   - **Users' Trajectory**: A map and some lines are used to show the trajectory of a specific user between any time.
   - **Heatmap**: A heatmap is used to show the popularity of specific POIs.
5. **Prediction-related Analysis**
   - **POI's visiting number next month**: Giving a specific POI_ID, the predicted visiting number of next month will be shown.
   - **User's next POI destination**: Giving a specific User_ID, the predicted next POI destination will be shown.

2. Implement Details

2.1 Data Structure Behind

I mainly use two vectors, two hash tables and a KDtree as my data structure.

- **Vector**: The first vector is used to store all the checking-in data, including user_ID, POI_ID and time. The second vector is used to store all the location information (i.e. latitude and longitude), which can be visited by POI_ID in O(1) time.
- **Hash Table**: The first hash table is used to find the starting index of a specific user in the checking-in data vector. The second hash table is used to find the index of a specific POI in the checking-in data vector. Both hash tables enable searching a specific user or POI in O(1) time.

- **KDtree**: A KDtree is used to allow a quick search of POI_IDs within a given GPS range. Instead of traversing all the checking-in data vector, which can take O(n) time, my KDtree allows a O $(\sqrt{n})$ search.

2.2 Multi-thread

To avoid being stuck when loading such a huge csv file, I add a new thread when loading file. As a result, we can feel free to drag the window or do some typing while loading.

Fig1. Progress Bar

2.3 Well Robust

I add all the input checks that I can think of, including but not limited to:

- Wait until all the file is loaded.
- Check whether the starting time is earlier than the ending time.
- Check whether the starting longitude/latitude is smaller than the ending longitude/latitude.
- Only correct number can be typed into POI_ID and user_ID input boxes.
- When there is no data available, notify the user.

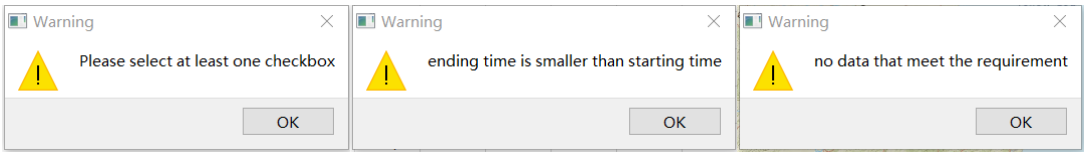

Fig2. Well Robust

2.4 Various Display Methods

- **QML Map**: QML (Qt Modeling Language) is a user interface markup language. I use QML to display markers, users' trajectory and the heatmap. Meanwhile, C++ is used to handle the processing of data and then transfer them to the QML file.
- **TextBrowser**: *Filter* and *Prediction Analysis* functions are simply shown by a text browser.
- **Line Chart**: Line charts are used to display *checking-in number over time* and compare *DAU of two POIs*.
- **Table**: A table is used to display the *top 5 POIs visited* by specific users every month.

2.5 Prediction

In prediction part, I have accomplished two functions:

- **POI's visiting number next month**: Given a specific POI_ID, I simply use the average visiting number of previous months as the prediction for next month. I find such prediction make sense, because judging by the *Checking-in Number* function, POI_ID's visiting number every month does not show much regular pattern.
- **User's next POI destination**: Users are more likely to go to the places that they recently visited. Based on this idea, the more recently visited, the higher weight the POI_ID will get. Meanwhile, the POI_IDs which are visited one month ago will be simply disgarded.

3. Results

Here are some pictures of my program. As is shown, there is a navigation bar on top of the program. Users can easily choose the function they want. Due to the limitation of page numbers, I will not introduce each picture in detail.

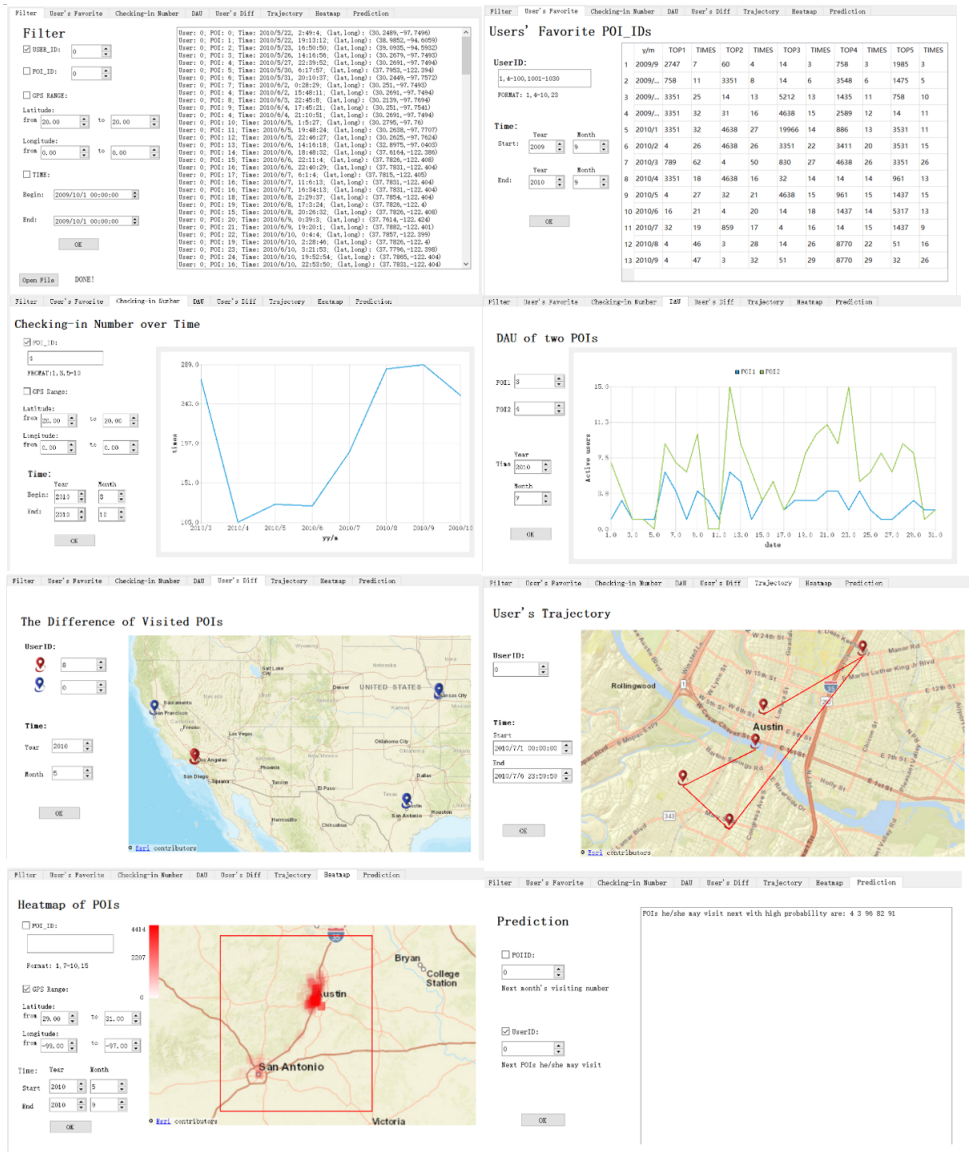

Fig3. Pictures of My Program

4. Discussion

I'd like to share something interesting I found from the data.

4.1 Population Distribution of America

With the *Heatmap* function, I draw the heatmap of the United States from 2010/1/1 to 2010/9/30. I find that the heatmap is very similar to the population distribution of USA. From the picture below, we can find that the northeast (the northeast coast and the Great Lakes region) is densely populated, the south and west coasts are more populous and the vast inland areas are sparsely populated.

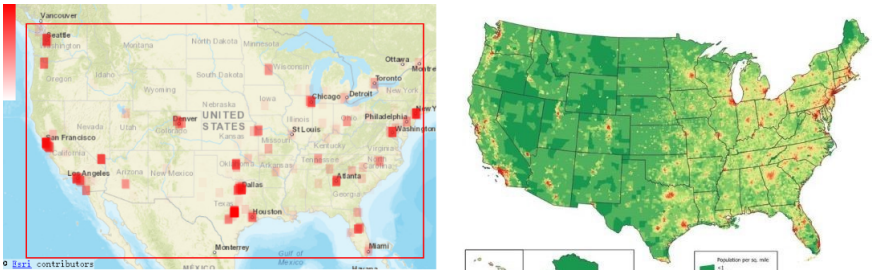

Fig4. Population Distribution of America

4.2 Satellite City

With the *Heatmap* function, I draw the heatmap around Los Angeles from 2009/10/1 to 2010/5/31. I find that not only Los Angeles but also many cities around it have a high POI visiting number, including Anaheim, Pasadena, Beverly Hills, Santa Monica and Glendale. These cities are known as satellite cities, whose emergence is the result of the expansion of central cities.

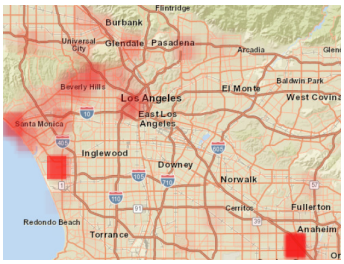

Fig5. Satellite Cities around LA

4.3 Checking-in Time ≠ Local Time

When I researched the checking-in time, I was surprised to find that much checking-in time was around 0am. However, when I drew the line chart of POI3's checking-in number every hour in a day (2010/8/9 Fig6 left) and compared it to its actual location (Fig6 right), I found there must be wrong with the checking-in time.

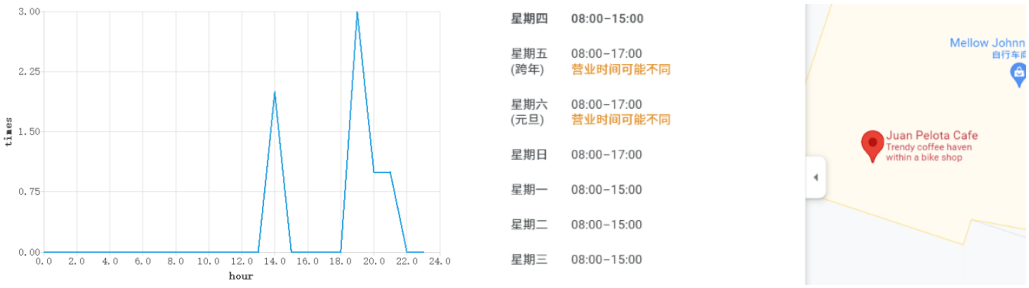

Fig6. POI3 checking-in number in 2010/8/9 (Monday) and its' opening time from google map

As is shown in Fig6, POI3 is a coffee shop opening from 8am to 3pm in Monday. How could the coffee shop be visited in 2, 19, 20, 21pm when it had been already closed? The answer is that checking-in time we get is UTC (Coordinated Universal Time) while local time should be Central Standard Time (CST) which is 6 hours behind UTC. When we subtract 6 from the time we get, the first checking-in is at 8am and the last is at 3pm, which is in good agreement with the actual open time.

4.4 Checking-in Time and Number Imply a Lot

Checking-in Number and *DAU* are two functions that mainly deel with the checking-in time and number. With help of these two functions, we can compare the popularity and even infer what kind of places it is. Apart from analysizing checking-in time hourly like what I do in chapter 4.3, here is another example.

I draw the DAU of POI3 and POI13 in 2010/8 (Fig7). As can be seen in Fig7, these two POI_IDs show a clear difference. POI13's DAU is much higher than POI3, meaning that POI13 should be a popular place like train stations, places of interest and large office buildings while POI3 is just a common place. If we find the actual location of POI3 and POI13, we will see that POI3 is a small coffee shop nearby a street while POI13 is Dallas/Fort Worth International Airport, the largest and busiest airport in Texas.

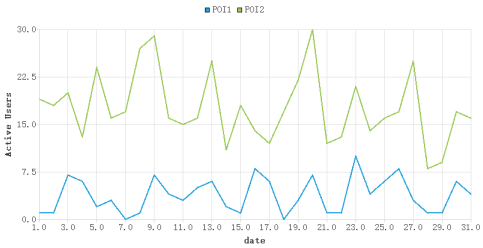

Fig7. DAU of POI3(blue) and POI13(green)

	y/m	TOP1	TIMES	TOP2	TIMES	TOP3	TIMES	TOP4	TIMES	TOP5	TIMES
1	2010/1	2916	7	68	6	5723	3	4518	3	2042	2
2	2010/2	4518	10	2916	7	68	3	2984	2	6171	2
3	2010/3	2916	10	4518	9	68	8	2939	2	5723	2
4	2010/4	2916	13	4518	9	68	7	5723	3	4616	2
5	2010/5	2916	7	68	6	4518	5	2939	2	4547	1
6	2010/6	68	9	2916	8	4518	6	5723	2	18503	1
7	2010/7	2916	12	68	3	4518	3	2939	2	4610	2
8	2010/8	2916	10	68	9	4518	4	18505	3	621	2
9	2010/9	2916	12	68	10	4518	3	69	1	584	1

Fig8.User429's Favourite

4.5 Users' Habits Can be Learnt

User's Favourite, *Users' Difference* and *Trajectory* are three functions deeling with users. We can get plenty of information with the help of these three functions. Here is an example about User429.

From Fig8 above, we can see POI2916, 4518 and 68 are three most visited locations by user429. From Fig9, we can see that both POI2916 and POI68 are visited around 8am. (Recall that in chapter4.3 we mention that checking-in time isn't local time. Also, the reason why the checking-time is changed from 14pm to 13pm is that they use winter time in early May but later turn to summer time.) If we take a closer look, we will find that both checking-ins take place on working days and occur alternately. Check their actual places on a map, and we will see POI2916 is *Filling Station Coffee - Midtown* and POI68 is *Filling Station Coffee - Westport*. I have strong reason to guess that user239 may be a worker at *Filling Station Coffee* or a big fan of it.

POI4518 seems to be a shopping mall. With the *Filter* function, I find that User5041 also visited this place from time to time. From Fig10 left, we can see there is overlap between two users. If we are making some platforms like *Douban* or *Dianping*, we can recommend user429 some places to go according to the places user5041 likes.

As for the trajectory, I can hardly find anything valuable due to too few daily data. However, put it into a long time like a month, and we will see some users have a trip to another city (Fig10 right).

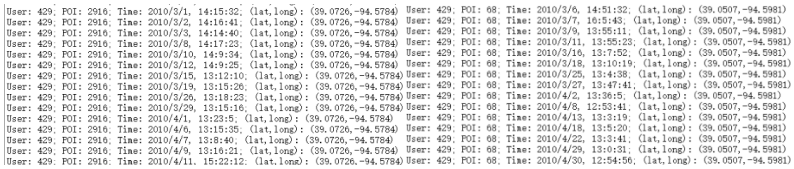

Fig9. Filter of User239 and POI68, 2916

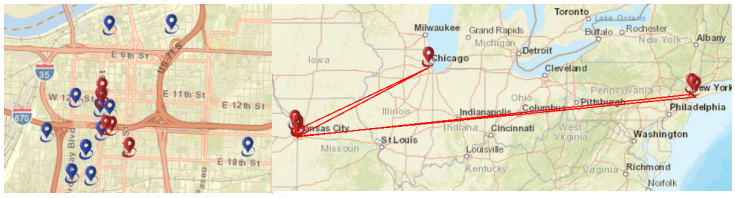

Fig10. Left: User239 and 5041; Right: User239 trajectory in 2010/1

5. Conclusion

I have used efficient data structures and plenty of display methods to build a powerful, user-friendly and robust app for data analysis and visualization of POI checking-in data. With the help of this application, recommendation platforms can better place their products, geographic workers can study geographic phenomena such as population distribution, and transportation hubs or businesses can take preventive measures against peak passenger flow.