Data Analysis Report for Indoor Location Data

(Project: A Machine Learning Based Indoor Positioning for IoT Systems)

Ho Yuan Ai

School of Information Technology, Monash University Malaysia

FIT3162

Dr. Tan Chee Keong

Oct. 17, 2020

1.0 Introduction

Since the emergence of Global Positioning System (GPS), location analytics has become a hot topic in data analytics. While location analytics has been done immensely in an outdoor environment, there has yet to be some benchmark techniques for indoor analytics in an indoor environment, mainly due to scarcity in data and privacy issues. This report summarizes all the modelling and analysis results associated with location analytics performed on the UjiIndoorLoc dataset (Torres-Sospedra et al., 2015). The dataset however has limited data, hence a comprehensive report of insights is unfeasible. Therefore, the main purpose of this report is to document the modelling or analysis techniques used, so as to demonstrate the usage of location analytics in helping to enforce social distancing. This report also serves to verify and showcase the usage of location analytics technique used on indoor location data, previously used for GPS location data. Outdoor location analytics and indoor location analytics are similar in many ways, as indoor spatial data can be perceived as a smaller scale of outdoor spatial data. Some of the main differences between outdoor location analytics and indoor location analytics are different ways of spatial modelling and different types of spatial context.

The UjiIndoorLoc dataset is an indoor localization dataset, which was officially used by the 2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN) for IPIN2015 competition. The dataset is focused on WLAN fingerprint positioning techniques. The location data was collected in a university campus in Valencia, Spain, and contains movement trajectories data of 19 users in multi-floor buildings. The dataset was chosen for analytics as users move around freely and the locations form meaningful trajectories.

The remainder of the report is organized as follows. Section 2 will describe the preliminary steps required before any useful analytics can be done on raw location data, in the form of longitude and latitude pairs. It includes information on setting up of system, resources required and techniques to transform raw data. Section 3 describes the implementation of the Coherent Moving Cluster (CMC) algorithm used to determine user clusters that moves together in close proximity. The results of this algorithm are used to warn users that are found in such clusters on the web application that has been developed. Section 4 presents the analytics of the location data in the spatial dimension, particularly to find areas of interest (or “hotspots”). Section 5 next presents the analytics of the location data in the spatio-temporal dimension. Section 6 then introduces the techniques used for feature extraction and similarity measure used for clustering users. Finally, section 7 summarises all the findings, insights, and presents some recommendations to enforce social distancing effectively. Note that section 2 to section 5 describes the methods and techniques used, while the results of analytics, with respect to social distancing, are presented in section 7.

2.0 Data Exploration and Transformation of Raw Location Data for Analytics

Raw location data, in the format of longitude and latitude pairs, associated with a user ID, provides limited information for fruitful analytics. Thus, in order to produce more useful results, some steps are done to offer more variability in representing a location. The universal way of storing and working with geometrical shapes, in the form of points, lines or polygons, is by using a data format called Shapefile. It was introduced by the Environmental Systems Research Institute (Esri). A shapefile not only contains locations, other data associated with each geometric feature can also be stored.

Shapefiles can be generated for shapes of buildings, shapes of rooms and even furniture. UjiIndoorLoc has metadata about the building number, floor and even space ID. Therefore, only one shapefile was generated for room shapes on a floor in one building, which acts as a showcase of how to obtain these data using the shapefile from raw location data. This can be done using geographic information system (GIS) software such as QGis, which is free and open-source, and ArcGIS, developed by Esri.

These GIS software usually provide interfaces to import and export data to PostgreSQL, which is the most popular database used for storing geometric shapes. PostgreSQL has an SQL extension, called PostGIS, to store and query geographic objects easily.

Figure 2.1

Shapefile created using QGis

Note. The floor plan image of the location for UjiIndoorLoc dataset was scraped from Google Maps.

Each room was assigned a self-defined room ID while the geometry shapes were created.

To demonstrate the usage of PostGIS to enquire the building and room (or space) number, an SQL script has been written to query the room ID of where each coordinates is in, and the mapped room ID is compared with the room IDs that are provided in the dataset. The results show that most self-defined room IDs can match with one and only one room ID; some deviations in rooms are because the geometry shapes of rooms were drawn manually. The full results can be found in the appendix of this report.

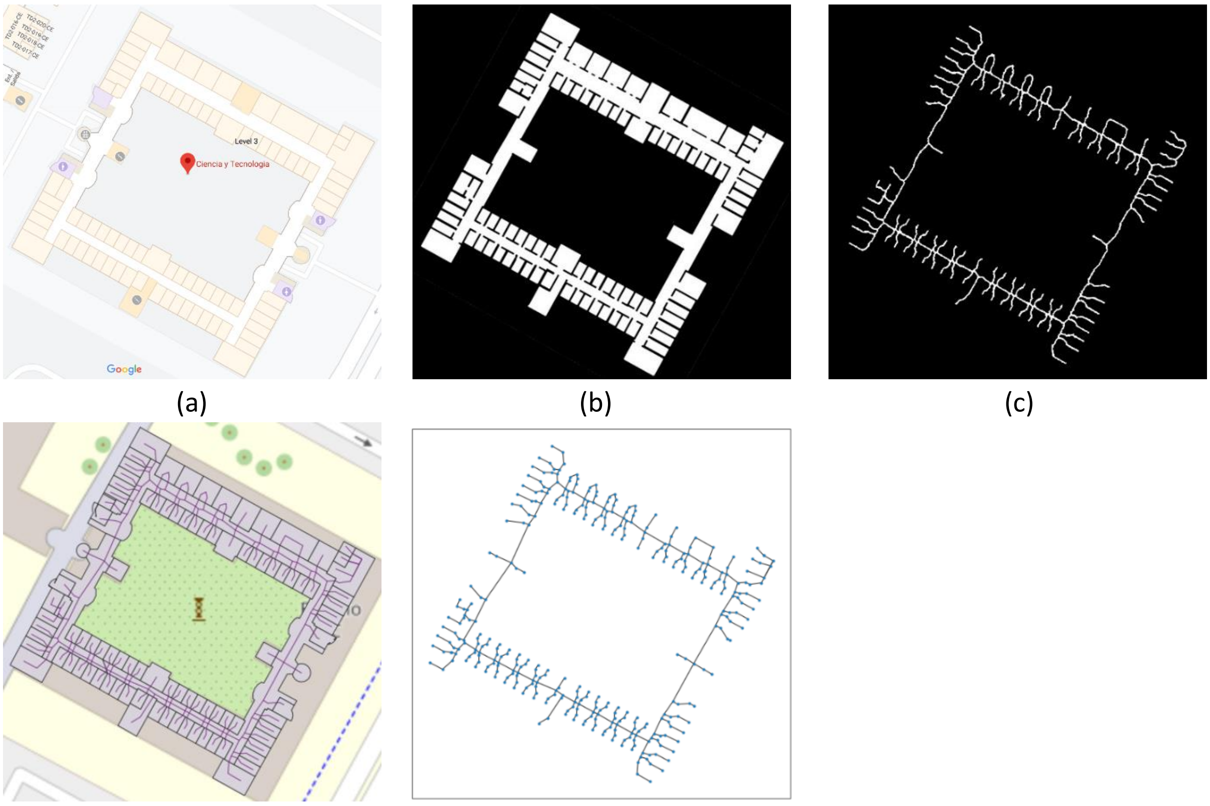
However, since the drawing of geometrical shapes of buildings and rooms are done before we train the model, this problem can be avoided unless the location predicted is inaccurate.

During the planning phase of the project, it was proposed to use a graph model to represent the geospatial topology of the indoor environment. It was found during the execution phase, that although the graph model is very useful especially for indoor wayfinding, it does not serve much purpose in the subsequent steps for analytics. This is because spatial context, which could be expressed more comprehensively using a grid-based model, is more useful in explaining the movement behaviours of users, rather than restricting the movement to predefined paths.

Nevertheless, a graph model was generated, which could be used for trajectory reconstruction, indoor wayfinding, network analysis. It was created by skeletonizing the floorplan using Lee's method (Lee, Kashyap & Chu, 1994). There are other ways in finding a suitable skeleton of the floorplan, such as Zhang's method of skeletonizing, medial axis transform and thinning. The output of Lee's method was used as it contains less short spurs, therefore can produce a simpler model. A binary image of the floorplan has to be created from the original floorplan, while the medial axis transform of the binary image was found using the Python library scikit-image and OpenCV. The edges were traced and drawn with the skeleton of the floorplan georeferenced on the map, and other two Python libraries, network and geopandas, are useful as a framework for the graph model and importing shape files respectively. The results of each process in the pipeline are shown in Figure 2.2.

Figure 2.2

Process of generating graph model. (a) Screenshot of floor plan; (b) Binary image of floor plan; (c) Output of skeletonization; (d) Georeferenced lines and generated shapefile; (e) Generated graph;



(a) (b) (c)
(d) (e)

3.0 Coherent Moving Cluster (CMC) algorithm

It was proposed that the Coherent Moving Cluster (CMC) algorithm be used to detect cluster of users (2 or more people) moving together for at least some time. The algorithm was proposed by Jeung et al. in 2008. An offline algorithm was first implemented, using the UjiIndoorLoc dataset as input.

The CMC algorithm takes a snapshot of the location of moving objects at certain points of time, then works only on the snapshot to find "convoys". While it keeps a copy of the convoys found in the last snapshot, it does not keep a copy of the last snapshot. In our implementation, the last snapshot is kept, for the purpose of retrieving the last detected location of some users which may be "lost", considering that Bluetooth energy is very volatile in an indoor environment. As a result, there will be a time buffer for lost beacons and unstable signal strengths.

In the implementation, snapshots of objects or users are taken at every fixed time interval, which was set to 5 seconds. During the time interval, multiple records of location of an object may be found, but only the last detected location will be used for detecting convoys. Another option is to calculate the average of all detected locations during the interval, but it will produce delayed results (since the timestamp of location detected is less recent while averaging out), and require more time to perform computations for averaging, affecting the performance if there are many objects.

Stricter conditions should be used for the definition of a convoy in an indoor environment, as there exist many obstructions which separates objects even though their geodesic distance with each other may be small. Other than the distance threshold proposed in the original CMC algorithm, it was imposed that objects must be in the same room to be in the same convoy. Therefore, one requirement of executing this implementation is to have the shapefile or database storing the geometry shapes of rooms ready.

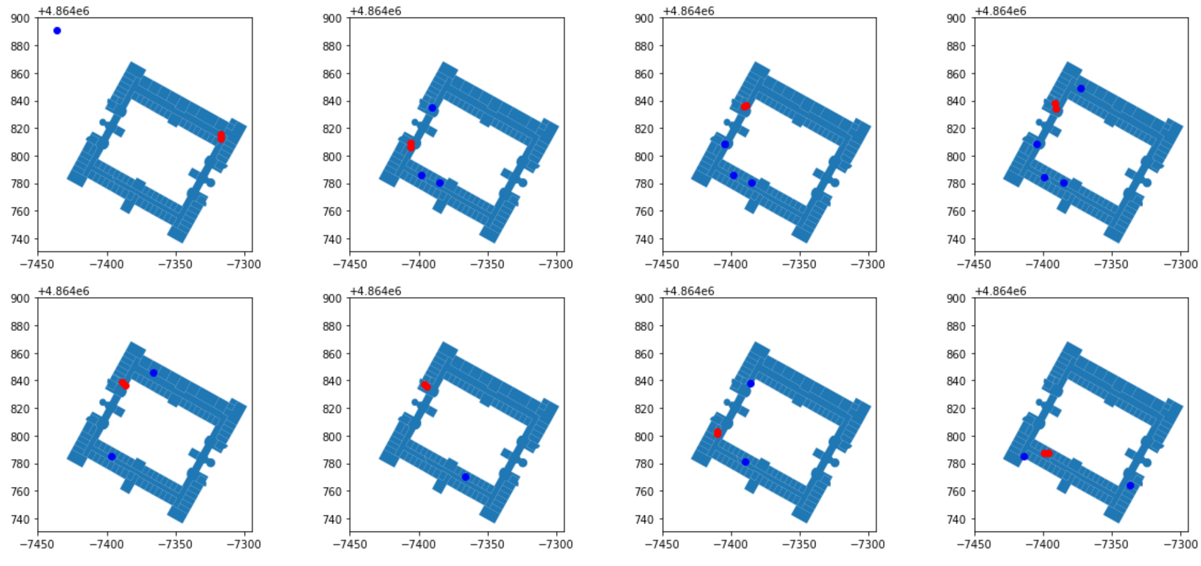
The metadata of each convoy found include the identifiers of objects in the convoy, the identifiers of records of location of objects detected from the database, along with the start time and the end time of the interval.

An undirected network consisting of users as nodes is created, where two users are connected with an edge if they are found in a common convoy. The edges are weighted by the total duration of the two users being found in the same convoy. The purpose of this network is to model the close contact of users using a network, so as to take precautions to prevent the transmission of virus between groups of people by keeping them apart. If a user is diagnosed with the virus, then this network is useful to identify the primary, secondary or even higher level of contacts of the user and notify them of such occasions.

According to the United States Centers for Disease Control and Prevention (2020), a close contact is defined as “someone who was within 6 feet of an infected person for at least 15 minutes starting from 2 days before illness onset until the time the patient is isolated”. Since the edge weights in the network is valued by the duration of contact, we can easily identify the list of users who are in “close contact” with a particular user.

Figure 3.1

Visualisation of some convoys detected in Building 2 Floor 3



4.0 Analytics in Spatial Dimension

Clustering of locations is done in an attempt to find areas of interest or "hotspots" in the entire area. The similarity measure used in this section is similar with the previous section, where locations in different rooms will not be in the same cluster. The clustering method that was used is Density-based Spatial Clustering (DBScan), as it can handle noisy data and ignores data points within scarce regions. Schubert et al. (2017) also argued that the DBScan algorithm performed competitively well, with suitable selection of hyperparameters. Konstantinidis et al. (2017) also uses DBScan to study the movements of elderly people in senior homes.

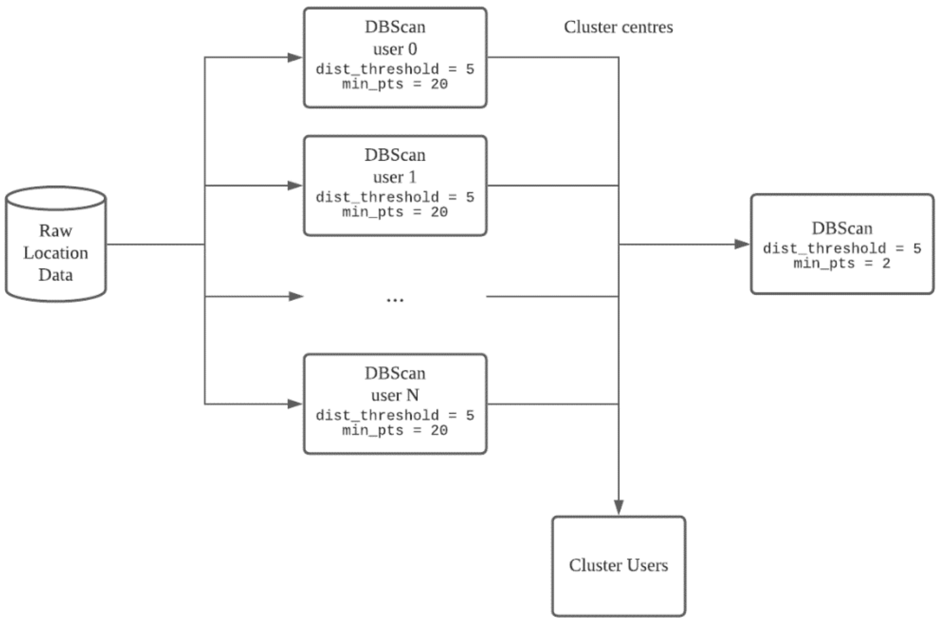
Note that DBScan may result in clusters with less data points than the hyperparameter defined while training with the data. This is because border points may belong to more than one cluster, and they will be assigned to only one cluster while producing the result (Schubert et al., 2017). One solution is to compute the results for different permutations of the input dataset, and verify the clusters by comparing the results. However, Schubert et al. advised that this precaution is not necessary, as it is very rare to encounter such cases.

While using the similarity measure mentioned above, we can observe that one cluster may be predominated by data points belonging to one user. This occurs when one user stays within a density-reachable area, resulting in numerous data points recorded within an area. When a second user is detected in the same area, then the new data point would be density-reachable from all data points of the first user within that area. This happens even if the second user only has one data point within that area. The clustering results obtained in this way would not be aligned with the initial aim of finding areas of interest or "hotspots", where there should be a number of distinct users detected in such areas.

Therefore, to improve the clustering results and acquire more meaningful clusters, the data points of each user are clustered separately first. Then, the cluster centres from the first clustering process will be used as input to the second clustering process, to discover areas where more than one user were detected. One advantage is that the results of the first clustering process might disclose some important behaviour of each user. Another advantage is that if a user does not stay at a spot for too long, such that the locations detected in some density-reachable area do not exceed the minimum points of a cluster, then these data points will be regarded as noise points in the first clustering process. This means that an area will not be identified as an area of interest, if there are many users passing by but do not remain in the area for too long. The figure below shows the flow chart of this clustering process.

Figure 4.1

Flow chart of DBScan clustering of location data points



For the UjiIndoorLoc dataset, if location points of all users are fed into the algorithm, then many clusters with only one data point are observed. This could be because the distance threshold is similar to the typical distances between location points. This problem is however resolved, when location points of each user are clustered separately first, then another clustering process is done on the multiple users' level. Although the distance threshold remains the same, many data points are removed during the first clustering process, thus removing boundary points.

Clustering was done using the whole dataset, since the size of the dataset is considered small. However, in practice, the number of data points grows incrementally each day, and it will be unfeasible to find clusters using every data point. Therefore, clustering can be done on a daily basis, weekly basis, or monthly basis, and this might reveal some temporal patterns. The data points in the UjiIndoorLoc dataset were mostly collected in one day, and therefore no patterns could be revealed.

5.0 Analytics in Spatio-temporal Dimension

In simpler terms, data in the spatio-temporal dimension can be described as trajectories, where location points associated with one object are linked together according to their time. To perform clustering on these trajectories, some important parameters need to be defined: what is the maximum time difference allowed and distance allowed between two trajectories, for them to be considered in the same cluster?

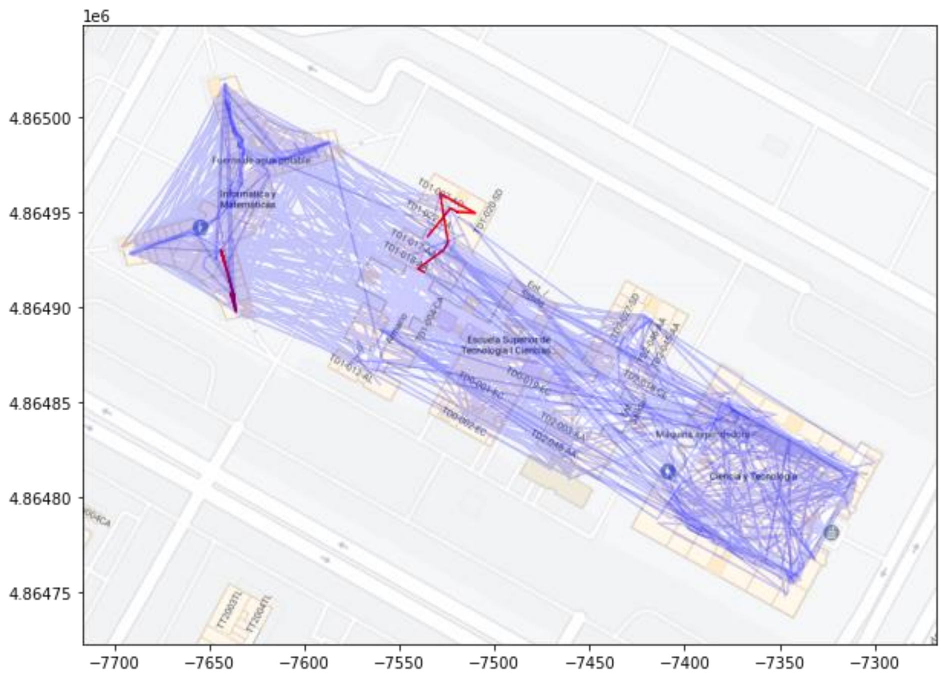
In an outdoor environment, trajectories clustering is often done to find a popular route from a fixed source to a fixed destination. This is especially useful in navigation, while giving route suggestions. While in an indoor environment, a similar approach can be done to assist in indoor wayfinding. This is out of the scope of this project, but it is possible for future work.

Edit distance on Real sequence (EDR) is used to measure the similarity between trajectories. It was proposed by Chen, Özsu and Oria in 2005. It is very similar with the original edit distance, but two location points in two sequences are considered the same if they are within a time difference and within a distance threshold defined. A time threshold of 3600 seconds (1 hour) and a distance threshold of 5 metres were used. Again, DBScan is used for clustering.

Although EDR is simple and effective to compute the similarity between two trajectories, its greatest disadvantage is that two trajectories are considered different if they do not start and end at similar locations. Consider all trajectories found in the dataset, as visualised in Figure 5.1, the trajectory of user 0 consists of large number of points, while other users have shorter trajectories. Even if some of these trajectories may overlap with each other, they would not be clustered in the same group, as a large amount of edit distance is imposed to "add" points to the shorter trajectory. Improvements need to be made to address these issues, by including characteristics of the Longest Common Substring problem while designing the algorithm.

Figure 5.1

Visualisation of Trajectories



6.0 Clustering Users

6.1 Feature Extraction and Preparation of Data

To be able to cluster users, some properties of their behaviours or movement patterns must first be extracted first, to be used as inputs for clustering algorithms. Features that have been extracted using the UjiIndoorLoc dataset are as follows:

1. The number of unique days of visit to the campus.
2. The average duration of visit to the campus.
3. The average duration spent in a building on different days.
4. The median time of day when a user visits the campus.

5. The number of times a user visits the campus on each day of the week (Monday to Sunday). There are 7 features extracted here, one for each day of the week.
6. The cluster centres found in section 4.0 in the first clustering process.

Since there are various types of features extracted, it is then very important to define the similarity measure of these features. Since features 1 – 5 can be considered as continuous data, their similarities are suitable to be found using their Euclidean distance. For feature 6, the cluster centres are first translated into text. For each cluster centre, one term in the form of building ID, floor number and space ID appended together is created. Then, these terms are perceived as words and the cosine similarity can be used as the similarity measure. In our implementation, the terms belonging to different users are converted to a matrix of term frequency-inverse document frequency (TF-IDF) features, where one user owns a document with the terms created as words. The output is then normalized, and Euclidean distance is used to measure the similarity between vectors.

Before clustering, the values of features are normalized for training, to prevent excessive effect of features with smaller variance to the clustering results. Normalization is used instead of standardization as normalization does not make any assumptions about the distribution of the features to estimate the mean and standard deviation of the population.

6.2 Clustering Methods

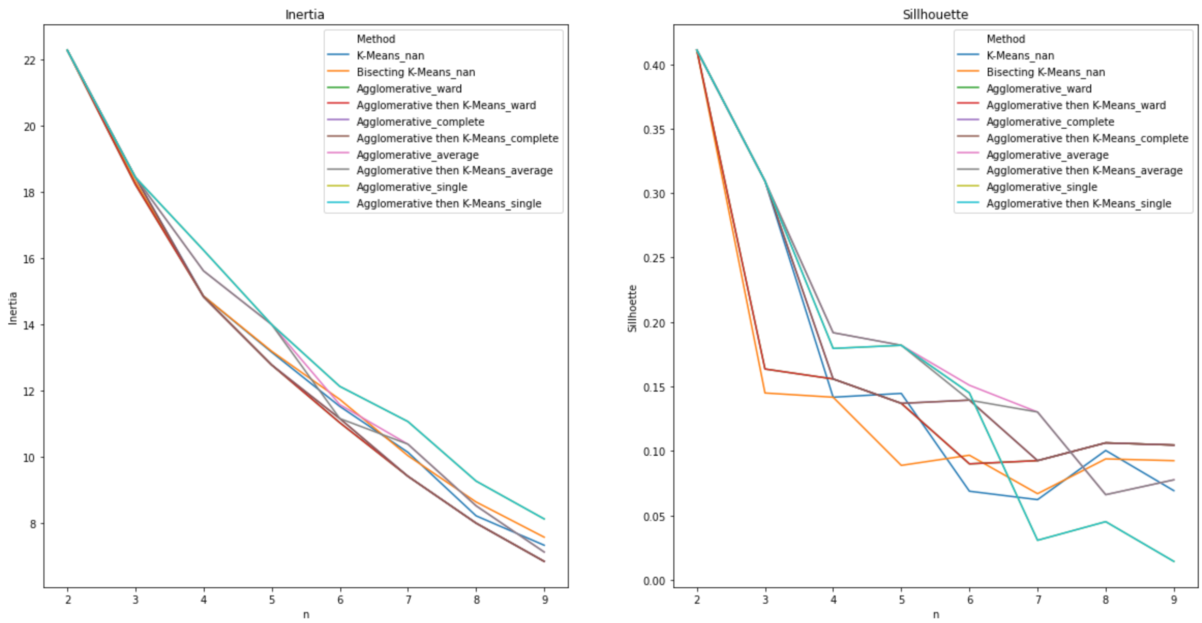
The clustering methods that were used for comparison include K-means, bisecting K-means and agglomerative clustering. K-means and agglomerative clustering from scikit-learn library (Pedregosa et al, 2011) were used, while bisecting K-means were implemented with a similar interface.

6.3 Clusters Evaluation

The results of each clustering method, using different values for the number of clusters, are compared using two metrics: inertia and silhouette score. The inertia is the sum of squared difference with cluster centres, while the silhouette score is the percentage difference of the mean intra-cluster distance and the mean nearest-cluster distance, according to the documentation of scikit-learn.

Figure 6.1

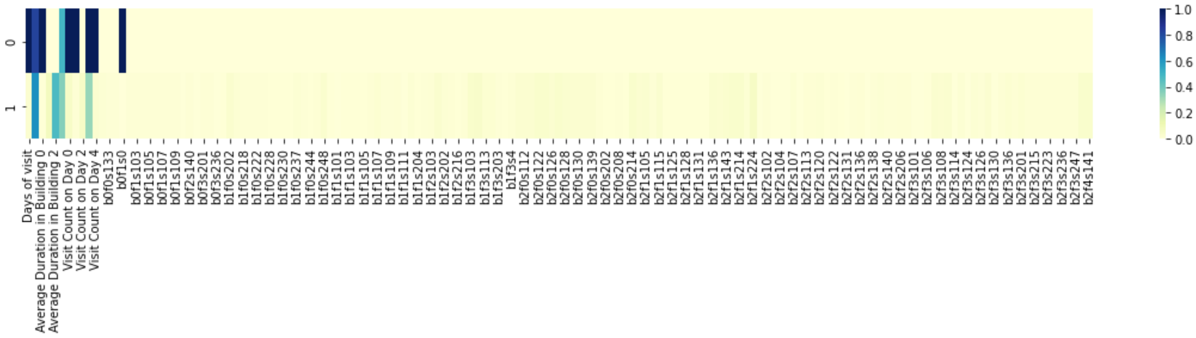
Graph of n against metrics score for each clustering methods



From figure 6.1, we can observe that the inertia score decreases with n. This is always the case, as when the number of clusters increases, some points may be removed from clusters to form a new cluster, and cluster centres will settle into a position nearer to each of the remaining points of a cluster. However, we can also observe that the silhouette score is optimum when n is 2 using whichever clustering methods. This is because silhouette score applies a penalty if a cluster consists of only one data point.

Figure 6.2

Heatmap of average values of each feature in clusters



When n is 2, all clustering methods cluster user 0 as different from all other users. In the visualisation above, User 0 is in cluster 0 and other users are in cluster 1. Looking at the visualisation in Figure 6.2, it is obvious that user 0 differs from the other user as it has the most data points, and its location points are found mostly in building 0, floor 1 and a space ID of 0. This is probably because this user has collected the most data points over a few days, but other users have only location records in one day. This difference is most probably not due to any user movement behaviour, but due to how much location points have been recorded. Another main difference is that all location points recorded by user 0 have a space ID of 0, which is inconsistent with location points of other users. This may be a flaw of the UjiIndoorLoc dataset.

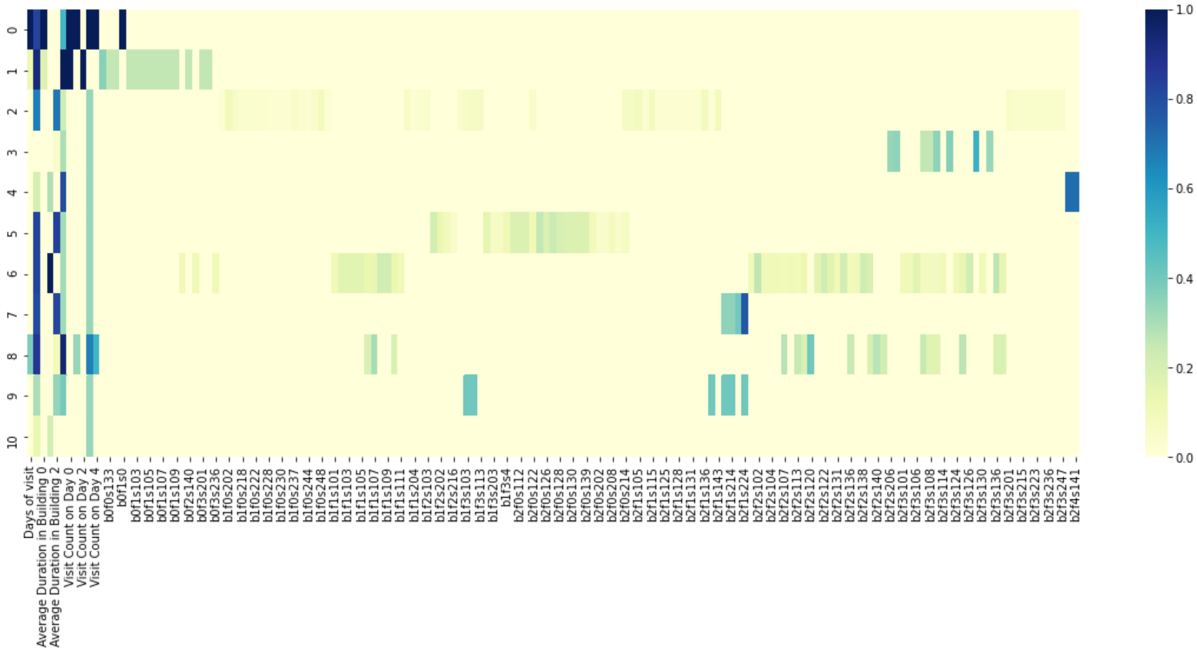
6.4 A Voting Mechanism for Clusters Evaluation

A voting mechanism was used to extract the most probable clusters. A network (or graph) with each user as one node was created to assist the mechanism. The mechanism works as such: For each cluster result obtained from Section 6.2, the weight of the edge connecting two users is increased by one if they are in the same cluster. The edges of the network thus encode the number of times a pair of users are in the same cluster. This network will be used in Section 7 for more analytics.

A threshold of 70 was used, so that a pair of users are considered in the same cluster only if they were in the same cluster in more than 70 models. There are two main clusters identified, Cluster 2 consisting of users 2, 5, 6, 12, 13, 16 and 17, and Cluster 5 consisting of users 8, 14 and 7. All other users were in their own clusters.

Figure 6.3

Heatmap of values of cluster centres



From figure 6.3, users in both Cluster 2 and Cluster 5 differ from other users as they stay in some distinct areas compared to other users. For example, users in cluster 5 are usually found in the second and third floor of Building 1, as well as the ground floor of Building 2. Other users are found in those areas.

7.0 Insights and Recommendations

7.1 Interpretation and comparison of outcomes

From section 3, it was shown that the CMC algorithm can be used to detect users close to each other for at least some time. Clustering using DBScan in section 3 is similar, as the CMC algorithm also uses DBScan. The main difference is that the CMC algorithm finds clusters at each snapshots of data points, while clustering in section 4 finds clusters on data points regardless of time. Consequently, the clusters found in section 3 reveal where and who stay close to each other at which point of time, while the clusters in section 4 reveal where users tend to stay.

Section 5 is another clustering method in location analytics. It is similar to the CMC algorithm in section 3, as both consider the time difference and location distances; they differ in the sense that clusters in section 5 do not necessarily mean the users are travelling at the same time, but in similar directions. On the other hand, clustering in section 5 is similar with section 4 as both find some "hotpots" or a popular stretch of area, but clustering in section 4 strips off the temporal dimension entirely.

Section 6 is straightforward, with users clustered into groups based on their movement behaviours.

7.2 Areas of interests

Using the cluster centres found in section 3 and in section 4, they can be labelled as "hotspots". Both find areas which are occupied by users frequently. More precautions should be done in these areas, such as:

1. Sanitation should be done more frequently in these areas.
2. The placement of sanitation materials, guides or signs to remind people to follow social distancing rules.
3. Locations of these crowded areas can be shown to users on the web application, so they can attempt to avoid these places if possible.

In addition to the above, clusters in section 4 also inform during what time of the day do users travel in those areas. Therefore, sanitation could be done during those times of the day, for more effective precautions.

The clusters found in section 3 and section 4 are shown in the visualization in figure 7.1.

Figure 7.1

Visualisation of areas of interest

7.3 Scheduling of Access to Campus

The networks found in Section 3 and Section 6 can be used to assist better planning of subsets of users to be allowed to be on the site on different days. Since the network from section 3 informs about the close contact of users, we can minimise the probability of contact between those that have not already had close contact with each other. This is to minimise the transmission of the virus to other users in the network, and to keep the size of connected components in the network to be small.

On the other hand, we should divide those with similar movement behaviours to have access to the site on different days. This is to lessen the traffic to even out the distribution of users across all locations.

This problem can be formalized into a scheduling optimization problem. The objective function of the problem can be expressed as follows:

1. Maximise the number of pairs of users who already have close contact with each other to have access to the site on the same days. The award added to the objective function is the normalized value of the duration of time the pair of users are in close contact, provided that these users are allocated on the same day (edge weight of network from Section 3).

2. Minimise the number of pairs of users who have the same movement behaviour to have access to the site on the same days. The penalty imposed on the objective function is the normalized value of the number of times the pair of users are grouped in the same cluster based on their movement behaviour (edge weight of network from Section 6).

Assume that only 70% of all users have access to the site each day, and each user is allowed to have access for exactly three days in a week. OR-Tools by Google provides a library for CP-SAT solver, which can be used to solve integer programming problems. The solver is used to solve the optimization problem formalized above, and an example of an optimal solution is shown in Table 7.1.

Table 7.1

Table of Scheduling of Access to Campus

Day	Monday	Tuesday	Wednesday	Thursday	Friday
Users allowed	User 4	User 0	User 0	User 0	User 1
	User 7	User 5	User 1	User 1	User 2
	User 8	User 6	User 2	User 2	User 3
	User 9	User 7	User 3	User 3	User 5
	User 15	User 9	User 4	User 4	User 6
		User 10	User 8	User 5	User 9
		User 11	User 10	User 6	User 10
		User 12	User 11	User 7	User 12
		User 13	User 12	User 8	User 13
		User 14	User 13	User 11	User 14
		User 15	User 16	User 14	User 15
		User 16	User 17	User 16	User 17
		User 18	User 18	User 17	User 18

8.0 References

Chen, L., Özsu, M., & Oria, V. (2005). Robust and fast similarity search for moving object trajectories. *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, 491–502. https://doi.org/10.1145/1066157.1066213

Contact Tracing. (2020, May 21). Retrieved from Centers for Disease Control and Prevention: https://www.cdc.gov/coronavirus/2019-ncov/php/contact-tracing/contact-tracing-plan/appendix.html#contact

Jeung, H., Yiu, M., Zhou, X., & Jensen, C. (2008). Discovery of Convoys in Trajectory Databases. arXiv.org.

Konstantinidis, E., Billis, A., Dupre, R., Fernández Montenegro, J., Conti, G., Argyriou, V., & Bamidis, P. (2017). IoT of active and healthy ageing: cases from indoor location analytics in the wild. *Health and Technology*, *7*(1), 41-49. doi:10.1007/s12553-016-0161-3

Lee, T., Kashyap, R. & Chu, C. (1994). Building Skeleton Models via 3-D Medial Surface Axis Thinning Algorithms. CVGIP. *Graphical Models and Image Processing*, *56*(6), 462–478. https://doi.org/10.1006/cgip.1994.1042

Pedregosa, Varoquaux, Gramfort, Vincent, Thirion, Grisel, Blondel, Müller, Nothman, Louppe, Prettenhofer, Weiss, Dubourg, Vanderplas, Passos, Cournapeau, Brucher, Perrot, & Duchesnay. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research, 12,* 2825-2830.

Perron, L. & Furnon, V. (2019, July 19). *OR-Tools* (Version 7.2) [Software Library]. Google. https://developers.google.com/optimization/

Schubert, E., Sander, J., Ester, M., Kriegel, H., & Xu, X. (2017). DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Transactions on Database Systems (TODS)*, *42*(3), 1–21. https://doi.org/10.1145/3068335

Torres-Sospedra, J., Rambla, Montoliu, Belmonte, & Huerta. (2015). UJIIndoorLoc-Mag: A new database for magnetic field-based localization problems. *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 1–10. https://doi.org/10.1109/IPIN.2015.7346763

Appendix

Table of self-defined room ID matched with that provided by the dataset

Self-defined room ID	Matched room ID from dataset
1	{106}
5	{102}
7	{101}
8	{140}
11	{201,202}
13	{203}
21	{214}
23	{216}
32	{230}
35	{236}
37	{0,240,241}
38	{0,242}
39	{243}
40	{0,244}
41	{0}
42	{245,246}
43	{247,248}
45	{0,136,137}
47	{0,134,135}
49	{132,133}
50	{131}
53	{130}
54	{0,129}
55	{128}
57	{109,110}
58	{108,109}
59	{107,108}
60	{0,107}
76	{125}
77	{126}
87	{223}
90	{215}
98	{0,101,102,103,104,105,111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,140,141,143,201,203,204,205,206,207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,250,253,254}

Note. Room 98 is mapped to many rooms in dataset as it is the corridor. The original dataset does not define an ID for the corridor but rather uses the ID of the closest room.

2.1 explore_transform_data

October 18, 2020

[3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[4]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) { return false; }
```

```
<IPython.core.display.Javascript object>
```

0.0.1 Load and Clean Data

[5]:
```
training = pd.read_csv('Data\\TrainingData.csv')
validation = pd.read_csv('Data\\ValidationData.csv')
combined = pd.concat([training, validation], axis=0)
combined.drop_duplicates(keep=False,inplace=True)
combined.to_csv('Data\\AllData.csv', index=False)
```

[42]:
```
# floorplan for visualisation
floorplan = plt.imread("Data\\UJI_B012_floorplan.png")
```

0.0.2 Explore Data

[6]:
```
combined.groupby(['FLOOR', 'BUILDINGID']).size()
```

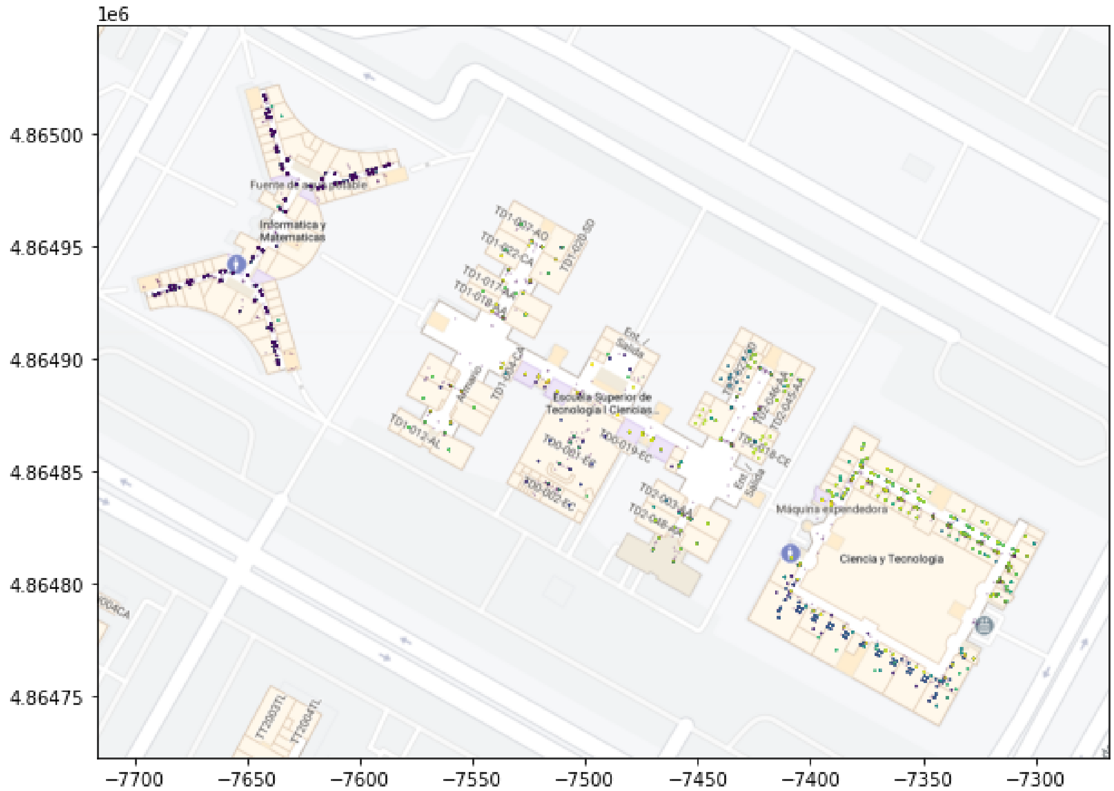
[6]:
```
FLOOR  BUILDINGID
0      0             1133
       1             1354
       2             1966
1      0             1564
       1             1363
       2             2273
2      0             1608
       1             1483
       2             1631
3      0             1474
       1              986
```

```
        2              2747
4       2               704
dtype: int64
```

[7]:
```
combined.groupby(['USERID']).size()
```

[7]:
```
USERID
0      1111
1      2731
2      1091
3       192
4       374
5       610
6       974
7      1383
8       507
9      1064
10      913
11     4516
12      437
13      108
14     1596
15      498
16     1024
17      717
18      440
dtype: int64
```

[53]:
```
x_left, y_bottom = -7717, 4864723
width, height = floorplan.shape[1], floorplan.shape[0]
scale = 0.40

plt.figure(figsize=(10,10))
plt.axes().set_aspect('equal', 'box')
plt.imshow(floorplan, extent=[x_left, x_left + scale*width, y_bottom, y_bottom␣
 ↪+ scale*height])
plt.scatter(combined['LONGITUDE'], combined['LATITUDE'], c=combined['USERID'],␣
 ↪alpha = 0.4, s=0.1)
plt.show()
```


1e6
4.86500
4.86495
4.86490
4.86485
4.86480
4.86475
−7700
−7650
−7600
−7550
−7500
−7450
−7400
−7350
−7300
Informatica y Matematicas
Escuela Superior de Tecnologia i Ciencias
Máquina expendedora
Ciencia y Tecnologia

2.2 generate_graph_model

October 18, 2020

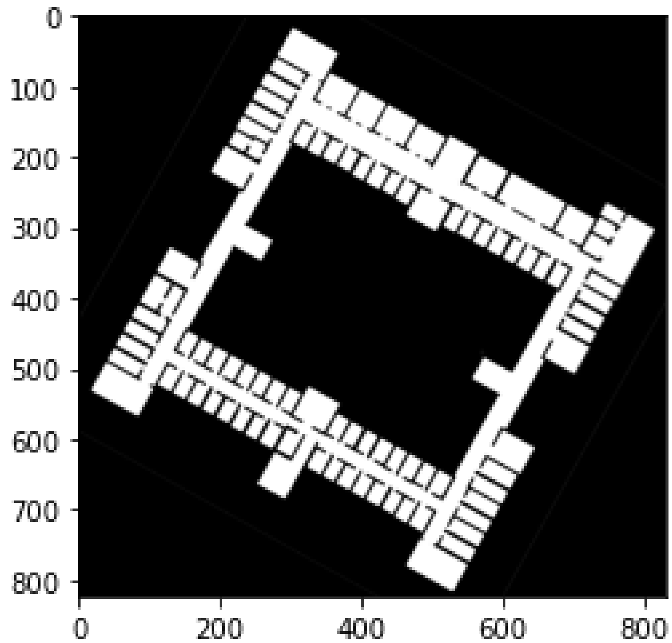
[1]:
```
import cv2
from skimage import color, morphology
import utils
from networkx import read_shp, DiGraph
import networkx as nx
import geopandas as gpd
from matplotlib import pyplot as plt
```

[13]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) { return false; }
```

```
<IPython.core.display.Javascript object>
```

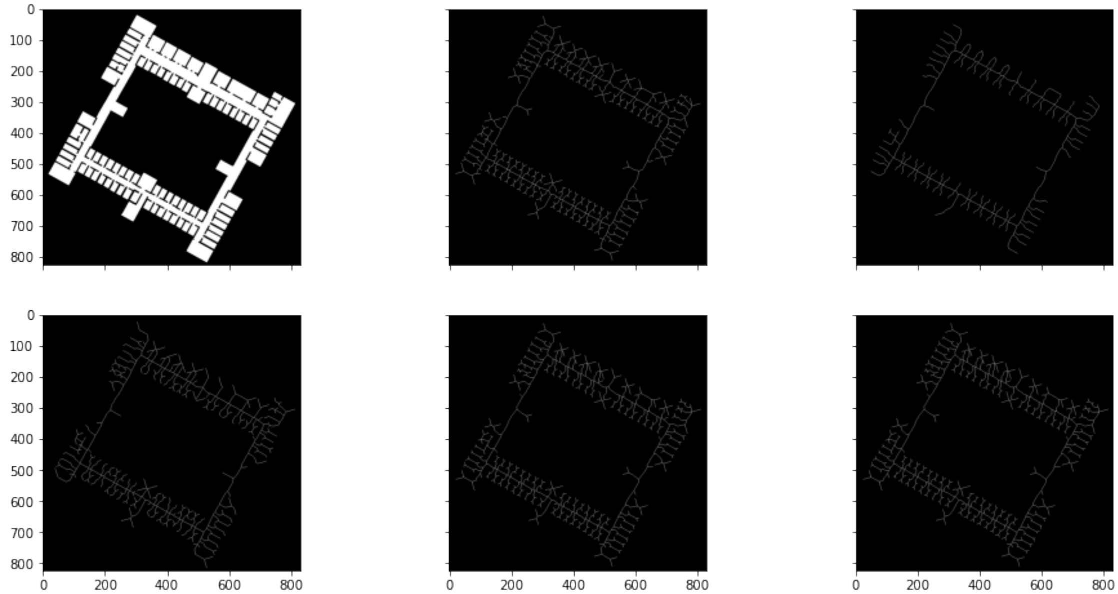
0.0.1 Find skeleton of binary floorplan

[2]:
```
b2f3_outline = cv2.imread('Data\\UJI_B2F3_binary.jpg',0)
plt.imshow(b2f3_outline, cmap=plt.cm.gray)
plt.show()
```



[3]:
```
fig, axes = plt.subplots(2, 3, figsize=(16, 8), sharex=True, sharey=True)
ax = axes.ravel()

bw_b2f3 = b2f3_outline > 195 * 1
ax[0].imshow(bw_b2f3, cmap=plt.cm.gray)

skel = morphology.skeletonize(bw_b2f3)
ax[1].imshow(skel, cmap=plt.cm.gray)

skel_lee = morphology.skeletonize(bw_b2f3, method='lee')
ax[2].imshow(skel_lee, cmap=plt.cm.gray)

axis, distance = morphology.medial_axis(bw_b2f3, return_distance=True)
ax[3].imshow(axis, cmap=plt.cm.gray)

thinned = morphology.thin(bw_b2f3)
ax[4].imshow(thinned, cmap=plt.cm.gray)

thinned_partial = morphology.thin(bw_b2f3, max_iter=30)
ax[5].imshow(thinned_partial, cmap=plt.cm.gray)
```

[3]:
```
<matplotlib.image.AxesImage at 0x265ccea6580>
```



This skeleton is then used to create shapefile using Qgis, by tracing on the georeferenced skeleton image.

0.0.2 Create graph model

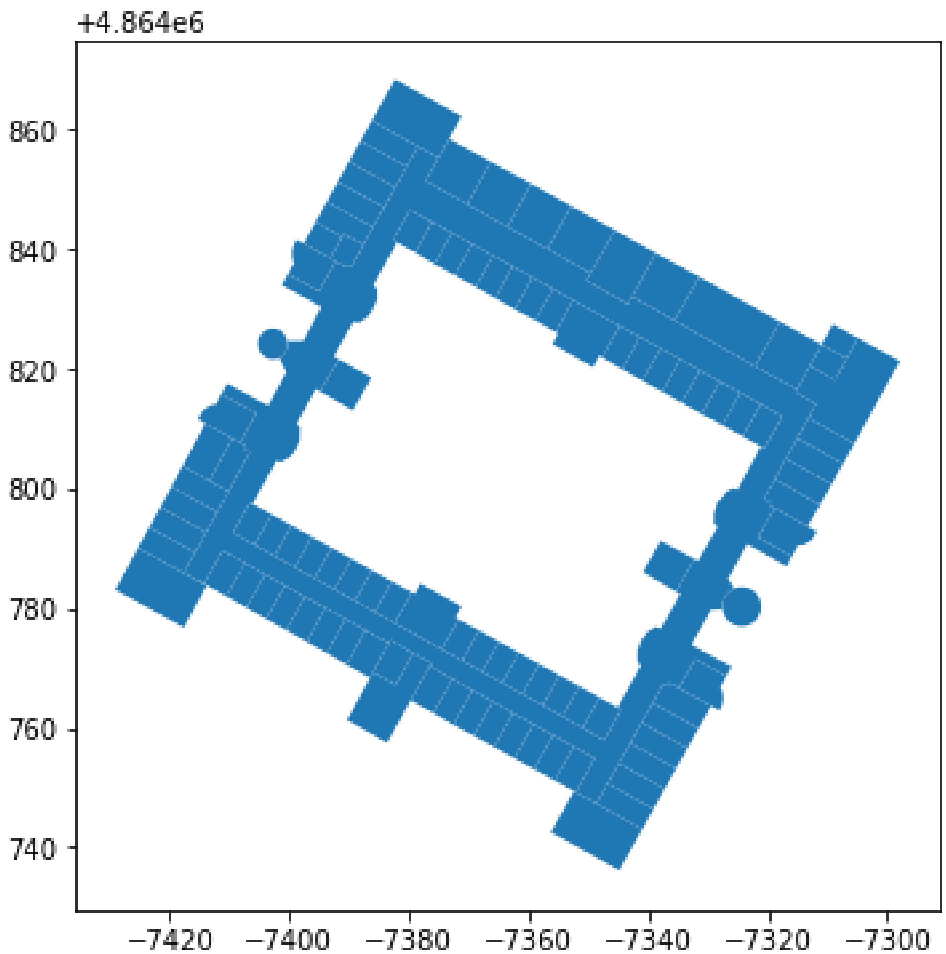
[4]:
```python
paths = read_shp("Outputs\\2\\UJI_B2F3_paths.shp", simplify=False).
↪to_undirected()
```

[14]:
```python
rooms = gpd.read_file("Outputs\\2\\UJI_B2F3_rooms.shp")
rooms
```

[14]:
```
     id  room_type                                           geometry
0     1  classroom  POLYGON ((-7382.330 4864868.381, -7371.186 486…
1     2  classroom  POLYGON ((-7386.200 4864861.438, -7386.200 486…
2     3  classroom  POLYGON ((-7388.082 4864858.060, -7380.367 486…
3     4  classroom  POLYGON ((-7390.013 4864854.595, -7386.148 486…
4     5  classroom  POLYGON ((-7391.884 4864851.214, -7384.154 486…
..  …        …                                                 …
93   95  classroom  POLYGON ((-7402.473 4864788.707, -7402.473 486…
94   96  classroom  POLYGON ((-7404.401 4864793.125, -7405.823 486…
95   97     stairs  POLYGON ((-7391.720 4864821.550, -7386.239 486…
96    6  classroom  POLYGON ((-7395.605 4864844.489, -7393.721 486…
97  100   corridor  MULTIPOLYGON (((-7372.330 4864835.615, -7372.3…

[98 rows x 3 columns]
```

[6]:
```python
buffer = []
for room in rooms.geometry:
    room2 = room.buffer(0)
    buffer.append(room2)

gpd.GeoSeries(buffer).plot(figsize=(6,6))
```

[6]: <AxesSubplot:>



Process and add useful data to edges

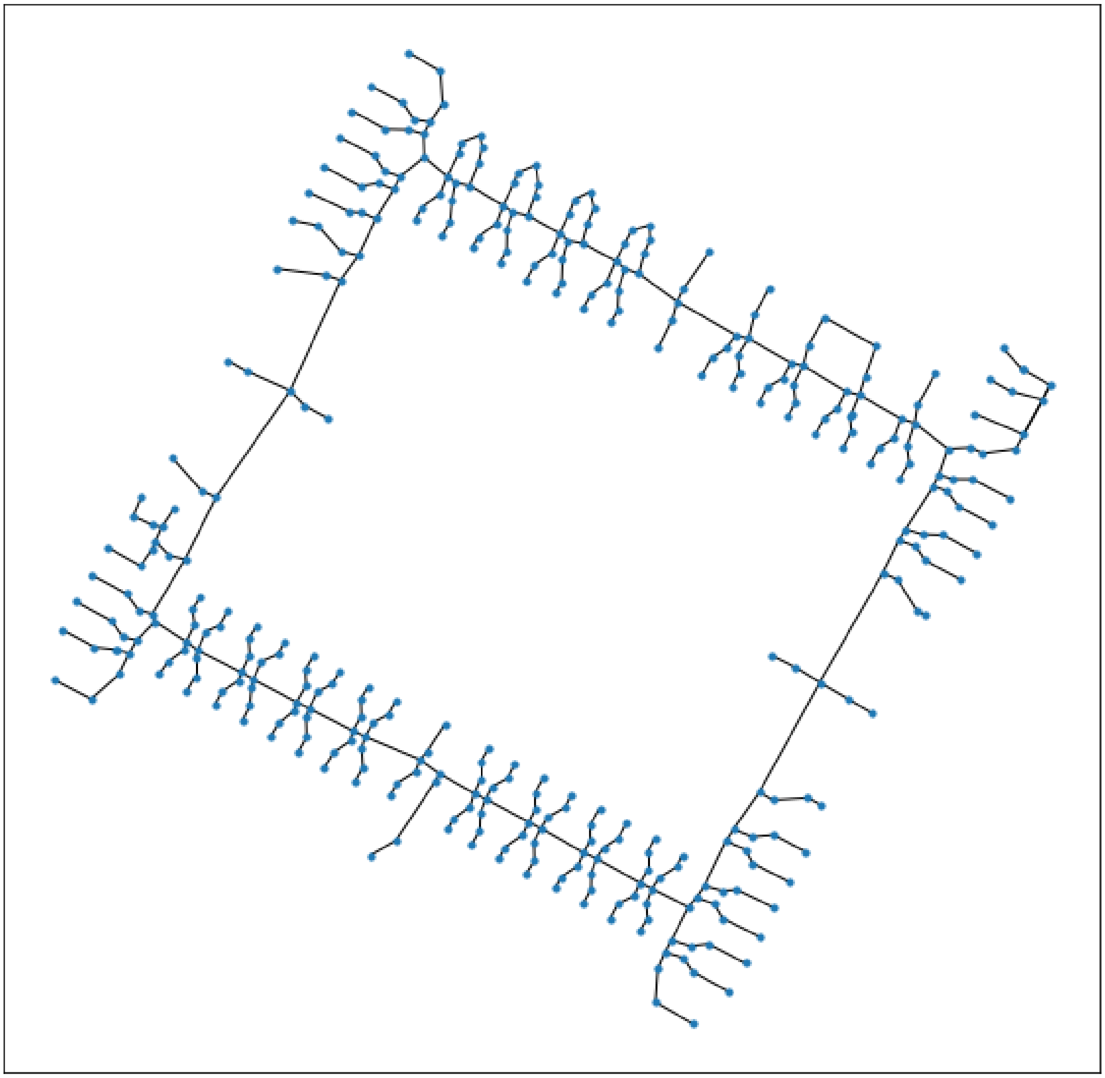
- id: a unique identifier
- length: spatial length of the edge in meters
- isEntrance: if the edge is a path to another floor
- isDoorway: if the edge is a path to another room

[8]:
```
# retrieve room id of entrances to floor
st_ids_arr = rooms[rooms['room_type'].isin(['lift','stairs'])]['id'].unique()

eid = 0
to_remove = []
for u, v, data in paths.edges(data=True):
    paths[u][v]['id'] = eid
    paths[u][v]['length'] = utils.geodesic_distance(u, v, is_3857=True)
    paths[u][v]['isEntrance'] = data['room_id'] in st_ids_arr
    if paths[u][v]['length'] < 0.000001:
        # remove edge from model
        to_remove.append((u,v))
    else:
        eid += 1

for r in to_remove:
    paths.remove_edge(r[0], r[1])
```

[18]:
```
isDoorDict = {}
for n in paths.nodes():
    adjacent_edges = paths.edges([n], data=True)
    room_ids = [e[2]['room_id'] for e in adjacent_edges]
    room_ids = list(set(room_ids))
    isDoorDict[n] = {'isDoorway': len(room_ids) > 1}

nx.set_node_attributes(paths, isDoorDict)
```

Visualize Graph Model

[16]:
```
plt.figure(figsize=(10,10))
nx.draw_networkx_nodes(paths, {n: n for n in paths}, node_size=10)
nx.draw_networkx_edges(paths, {n: n for n in paths}, width=1)
plt.show()
```



Write to output file to store the graph model

[12]:
```
nx.write_shp(paths, "Outputs\\2\\Graph Model")
```

3 CMC + Network Analysis

October 18, 2020

[1]:
```
import pandas as pd
import geopandas as gpd
from pyproj import Transformer
from geopy.distance import distance
from shapely.geometry import Point
from CoherentMovingCluster import CoherentMovingCluster
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
```

[2]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) { return false; }
```

<IPython.core.display.Javascript object>

[3]:
```
uji_data = pd.read_csv('Data\\AllData.csv')
```

0.1 CMC using only data in building 2 floor 3

The shapefile that defines the room geometries for building 2 floor 3 is used to determine the space ID for each coordinate. This is to demonstrate how CMC can be used in a real application.

[4]:
```
b2f3 = uji_data[(uji_data['BUILDINGID'] == 2) & (uji_data['FLOOR'] == 3)]
b2f3 = uji_data[['TIMESTAMP', 'USERID', 'LONGITUDE', 'LATITUDE']]
```

[5]:
```
room_gdf = gpd.read_file("Outputs\\2\\UJI_B2F3_rooms.shp")
cleaned_rooms = []
for room in room_gdf.geometry:
    room2 = room.buffer(0)
    cleaned_rooms.append(room2)
cleaned_rooms = gpd.GeoSeries(cleaned_rooms)
```

Define some hyperparameters and distance measure of data points for DBScan which is used in CMC.

[6]:
```
dist_threshold = 2
min_pts = 2
min_lifetime = 5
```

[7]:
```
TRANSFORMER = Transformer.from_crs("epsg:3857", "epsg:4326")

def row_geodesic_distance(row1, row2):
    row1_room = set(room_gdf[room_gdf.geometry.contains(Point(row1[2],
 ↪row1[3]))].id.tolist())
    row2_room = set(room_gdf[room_gdf.geometry.contains(Point(row2[2],
 ↪row2[3]))].id.tolist())

    if len(row1_room.intersection(row2_room)) == 0:
        # assume they are unreachable if coordinates are in different rooms
        return dist_threshold + 10

    x1, y1 = TRANSFORMER.transform(row1[3], row1[2])
    x2, y2 = TRANSFORMER.transform(row2[3], row2[2])

    return distance((x1, y1), (x2, y2)).meters
```

[8]:
```
cmc = CoherentMovingCluster(min_pts, min_lifetime, dist_threshold,
 ↪row_geodesic_distance)
convoys_list = cmc.offline_cmc(b2f3)
convoys = pd.DataFrame.from_records([c.to_dict() for c in convoys_list])
```

```
D:\USER\Anaconda\envs\geo_env\lib\site-packages\sklearn\utils\validation.py:67:
FutureWarning: Pass min_samples=2 as keyword args. From version 0.25 passing
these as positional arguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

Results

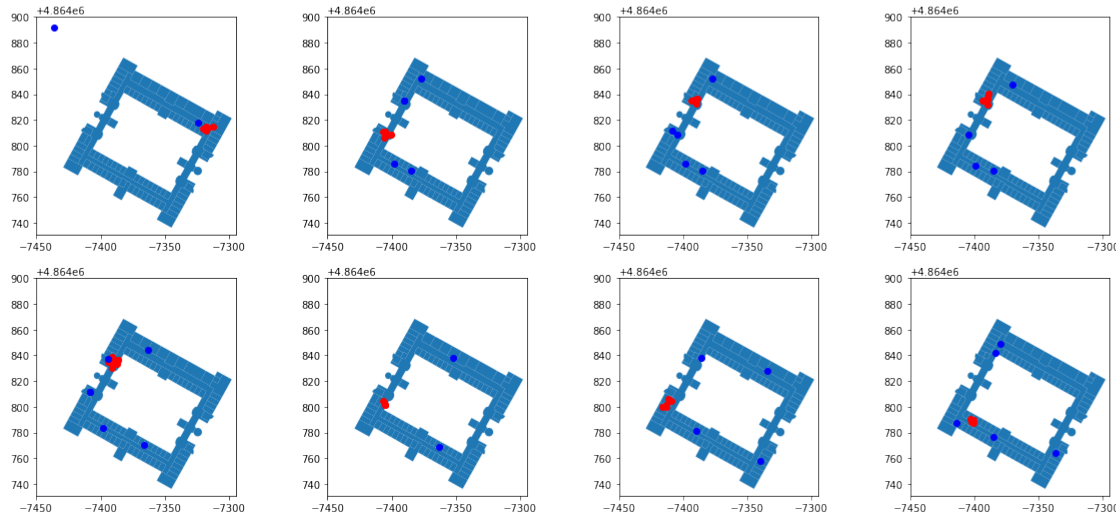
[9]:
```
convoys
```

[9]:
```
                                                oids  \
0                       [[12, 14], [12, 14], [12, 14]]
1                             [[8, 9], [8, 9], [8, 9]]
2                          [[5, 10], [5, 10], [5, 10]]
3                          [[5, 10], [5, 10], [10, 5]]
4   [[5, 17], [5, 17], [5, 17], [5, 17], [5, 17], …
..                                                 …
80                                [[7, 11], [7, 11]]
81                              [[11, 15], [11, 15]]
82                                [[7, 15], [15, 7]]
83                                [[7, 15], [15, 7]]
84      [[7, 15], [7, 15], [7, 15], [7, 15], [15, 7]]

                                                rids  start_time    end_time
0    [[11785, 12527], [11827, 12223], [11874, 12223]]  1371716399  1371716413
1          [[4695, 6127], [4853, 5692], [5054, 6020]]  1371719834  1371719848
2          [[1740, 7055], [1868, 6234], [1929, 6234]]  1371719839  1371719853
```

```
3          [[2230, 6417], [1671, 6785], [6232, 1671]]  1371719864  1371719878
4   [[1866, 17793], [1990, 17958], [2056, 18228], …  1371719894  1371719928
..                                                 …           …           …
80                         [[3641, 7543], [4192, 7543]]  1371723374  1371723383
81                      [[10231, 18118], [7544, 18118]]  1371723469  1371723478
82                       [[3376, 17866], [17952, 3376]]  1371724629  1371724638
83                       [[4624, 13830], [13854, 4624]]  1371724839  1371724848
84  [[3519, 13875], [3933, 13921], [4485, 13945], …  1371724889  1371724913

[85 rows x 4 columns]
```

[13]:

```
y_bottom, y_top = min(b2f3['LATITUDE']), max(b2f3['LATITUDE'])
x_left, x_right = min(b2f3['LONGITUDE']), max(b2f3['LONGITUDE'])
x_right += 5
y_bottom -= 15
x_left = -7450
y_top = 4864900

stdev = 0.01*max([y_top-y_bottom, x_right-x_left])
def rand_jitter(arr):
    return arr + np.random.randn(len(arr)) * stdev

plt.figure(figsize=(20,100))
size = 0
for i, convoy in convoys.iterrows():
    ax = plt.subplot(21, 4, i+1)

    all_locs = b2f3[(b2f3['TIMESTAMP'] > convoy.start_time) &␣
 ↪(b2f3['TIMESTAMP'] <= convoy.end_time)]
    all_locs = all_locs.loc[all_locs.groupby('USERID').TIMESTAMP.idxmax()]

    all_rids = [i for snapshot in convoy.rids for i in snapshot]
    convoy_locs = b2f3.iloc[all_rids]
    non_convoy_locs = all_locs.loc[~all_locs.index.isin(all_rids)]

    cleaned_rooms.plot(ax=ax)
    ax.set_ylim(y_bottom, y_top)
    ax.set_xlim(x_left, x_right)
    ax.scatter(rand_jitter(convoy_locs['LONGITUDE']),␣
 ↪rand_jitter(convoy_locs['LATITUDE']), color="r")
    ax.scatter(non_convoy_locs['LONGITUDE'], non_convoy_locs['LATITUDE'],␣
 ↪color="b")
    size += 1
    if size == 8:
        break

plt.show()
```



Output to store

```
[14]: convoys.to_csv("Outputs\\3\\B2F3_convoys.csv")
```

0.2 CMC using all data

The pre-defined space ID, Building ID, and floor that comes with the dataset are used here to find all convoys in the entire dataset.

```
[15]: all_data = uji_data[['TIMESTAMP', 'USERID', 'LONGITUDE', 'LATITUDE', 'FLOOR', 'BUILDINGID', 'SPACEID']]
```

Define some hyperparameters and distance measure of data points for DBScan which is used in CMC.

```
[16]: dist_threshold = 2
min_pts = 2
min_lifetime = 5

def row_geodesic_distance(x, y):
    if x[4] != y[4] or x[5] != y[5] or x[6] != y[6]:
        # assume they are unreachable if they are in diff building, floor or room
        return dist_threshold + 10

    lat1, long1 = TRANSFORMER.transform(x[3], x[2]) # lat, long
    lat2, long2 = TRANSFORMER.transform(y[3], y[2])

    return distance((lat1, long1), (lat2, long2)).meters
```

[17]:
```
cmc = CoherentMovingCluster(min_pts, min_lifetime, dist_threshold,␣
↪row_geodesic_distance)
convoys_list = cmc.offline_cmc(all_data)
convoys_all = pd.DataFrame.from_records([c.to_dict() for c in convoys_list])
```

```
D:\USER\Anaconda\envs\geo_env\lib\site-packages\sklearn\utils\validation.py:67:
FutureWarning: Pass min_samples=2 as keyword args. From version 0.25 passing
these as positional arguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

[18]:
```
convoys_all.to_csv("Outputs\\3\\all_convoys.csv")
```

0.3 Network Analysis of Users in Close Contact

Generate adjacency matrix then network of users according to how close they are to each other

[23]:
```
from itertools import combinations
users_count = uji_data['USERID'].unique().shape[0]
cmc_network = np.zeros(shape=(users_count, users_count))

for (_, data) in convoys_all.iterrows():
    all_oids = set([i for snapshot in data.oids for i in snapshot])
    for pair in list(combinations(all_oids, 2)):
        duration = sum([(pair[0] in snapshot and pair[1] in snapshot) for␣
↪snapshot in data.oids]) * CoherentMovingCluster.TIME_INTERVAL
        cmc_network[pair[0], pair[1]] += data.end_time - data.start_time
        cmc_network[pair[1], pair[0]] += data.end_time - data.start_time

cmc_network = pd.DataFrame(cmc_network)
gr = nx.from_pandas_adjacency(cmc_network)
```

Network Analysis

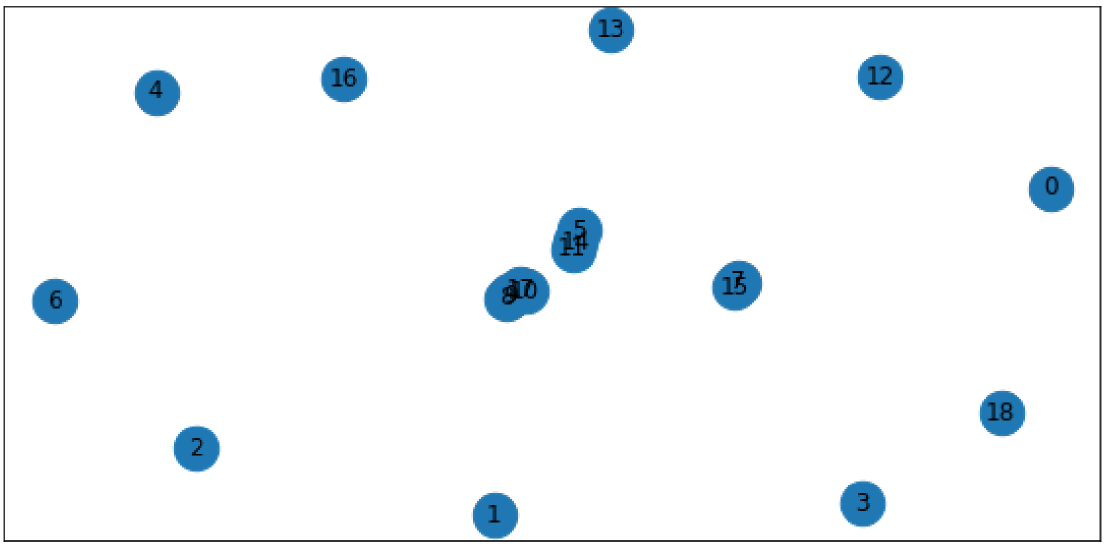
[24]:
```
for c in nx.connected_components(gr):
    print(c)
```

```
{0}
{1}
{2}
{3}
{4}
{11, 5, 14}
{6}
{15, 7}
{8, 9, 10, 17}
{12}
{13}
```

```
{16}
{18}
```

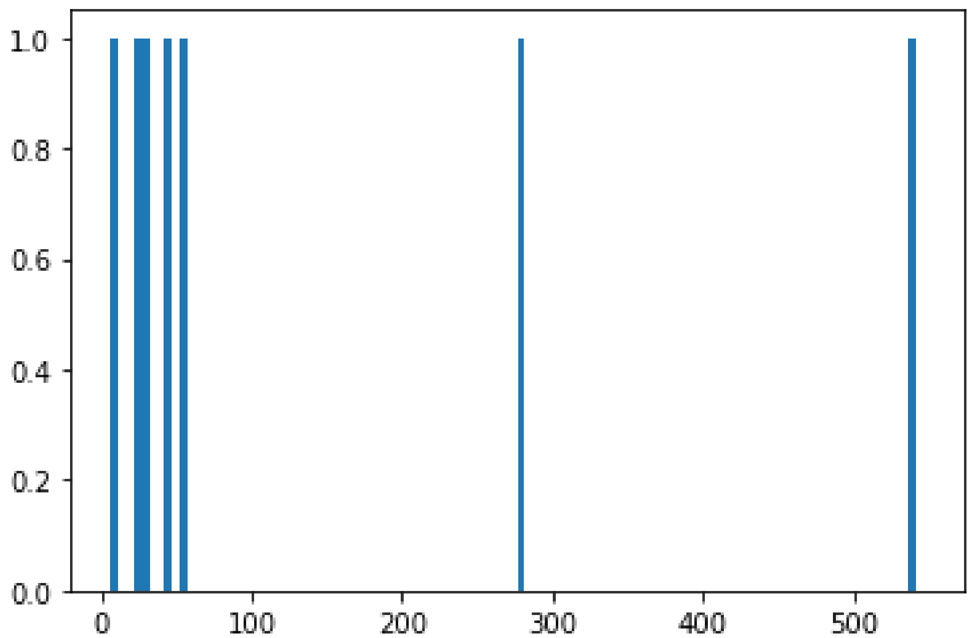
[25]:

```
import matplotlib.pyplot as plt
import networkx as nx

plt.figure(figsize=(10,5))
scaled_weights = [gr[u][v]['weight']/100 for u, v in gr.edges()]
nx.draw_networkx(gr, node_size = 500, with_labels=True, width=scaled_weights)
plt.show()
```



[26]:

```
# distribution of weights
weights = [gr[u][v]['weight'] for u, v in gr.edges()]
weights = np.array(weights)
d = np.diff(np.unique(weights)).min()
left = weights.min() - float(d)/2
right = weights.max() + float(d)/2
plt.hist(weights, np.arange(left, right+d, d))
plt.show()
```



Save to output file To be used in in "7 optimise_user_schedule"

[27]:
```
np.savetxt('Outputs\\3\\cmc_adj_mat.csv', cmc_network, delimiter=',')
```

4 DBScan

October 18, 2020

[2]:
```
from sklearn.cluster import DBSCAN
import pandas as pd
import geopandas as gpd
from pyproj import Transformer
from geopy.distance import distance
import numpy as np
from matplotlib import pyplot as plt
```

[1]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) { return false; }
```

```
<IPython.core.display.Javascript object>
```

[40]:
```
uji_data = pd.read_csv('Data\\AllData.csv')
dbscan_fit = uji_data[['TIMESTAMP', 'USERID', 'LONGITUDE', 'LATITUDE', 'FLOOR',␣
 ↪'BUILDINGID', 'SPACEID', 'RELATIVEPOSITION']]

# floorplan for visualisation
floorplan = plt.imread("Data\\UJI_B012_floorplan.png")
```

0.1 Clustering data points of each user

Define hyperparameter and similarity metric for DBScan

[5]:
```
dist_threshold = 5
min_pts = 20
TRANSFORMER = Transformer.from_crs("epsg:3857", "epsg:4326")

def similarity(x, y):
    if x[4] != y[4] or x[5] != y[5] or x[6] != y[6]:
        # assume they are unreachable if they are in diff building, floor or␣
 ↪room
        return dist_threshold + 10

    x1, y1 = TRANSFORMER.transform(x[3], x[2])
    x2, y2 = TRANSFORMER.transform(y[3], y[2])
```

```
    temp = distance((x1, y1), (x2, y2)).meters
    return temp
```

DBScan

[6]:
```
dbscan = DBSCAN(dist_threshold, min_pts, metric=similarity)
cluster_user = dbscan_fit.groupby(['USERID']).apply(lambda x: dbscan.
 ↪fit_predict(x))
```

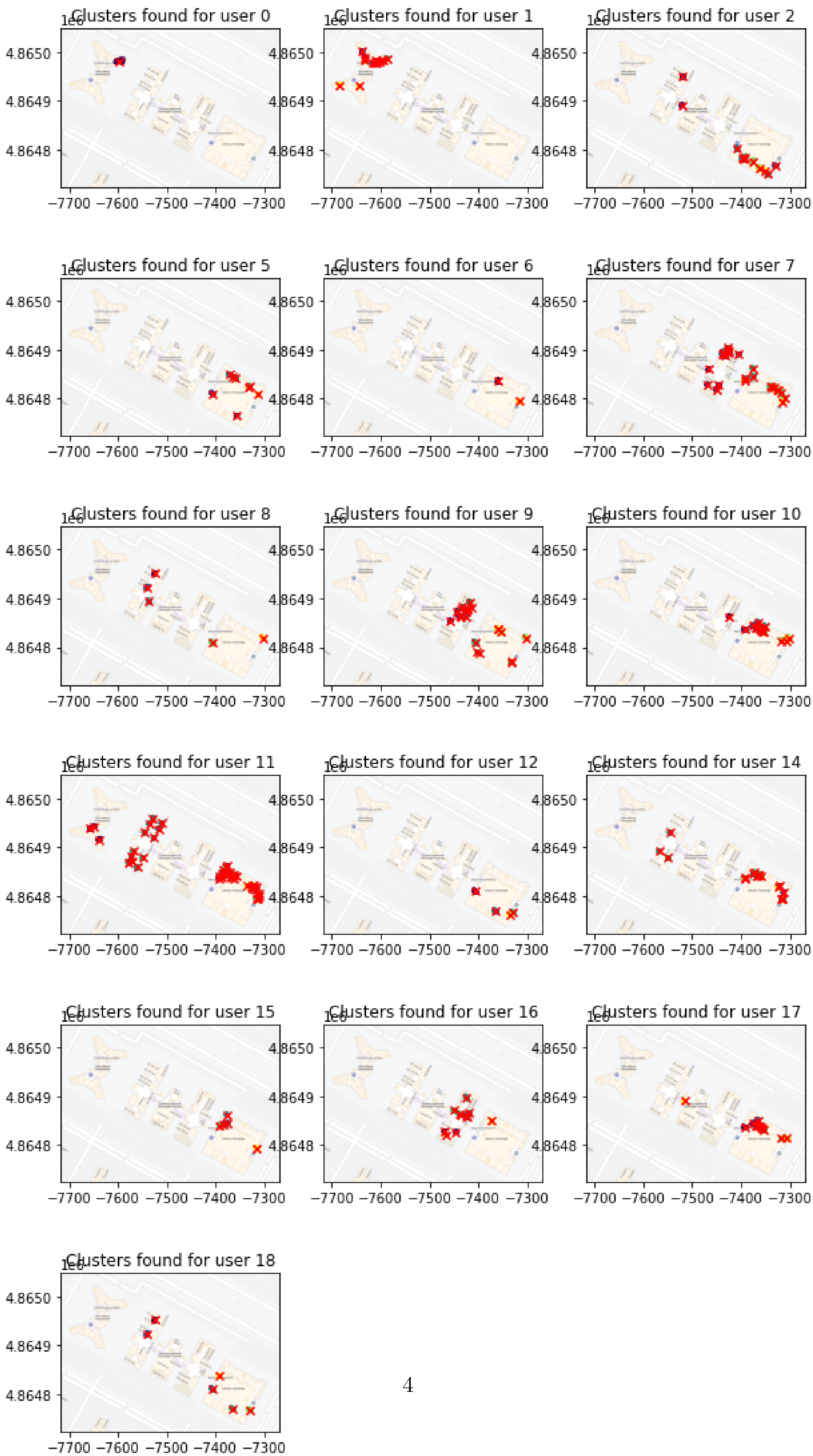
```
D:\USER\Anaconda\envs\geo_env\lib\site-packages\sklearn\utils\validation.py:67:
FutureWarning: Pass min_samples=20 as keyword args. From version 0.25 passing
these as positional arguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

[43]:
```
pd.options.mode.chained_assignment = None
dbscan_fit["cluster"] = -1

i = 0
for group, data in dbscan_fit.groupby(['USERID']):
    dbscan_fit.iloc[dbscan_fit['USERID'] == group, 8] = cluster_user[i]
    i+= 1
```

Visualisation

[45]:
```
x_left, y_bottom = -7717, 4864723
width, height = floorplan.shape[1], floorplan.shape[0]
scale = 0.40

plt.figure(figsize=(10,20))
i = 1
for group, data in dbscan_fit.groupby(['USERID']):

    removed_noise = data[data['cluster'] != -1]
    removed_noise = removed_noise[removed_noise['cluster'].
 ↪duplicated(keep=False)] # remove cluster with only one datapt

    if removed_noise.shape[0] == 0:
        continue

    ax = plt.subplot(6, 3, i)

    centroids_user = removed_noise.groupby('cluster')[['LONGITUDE',␣
 ↪'LATITUDE']].apply(lambda x: x.mean())
    centroids_user['cluster'] = centroids_user.index

    ax.imshow(floorplan, extent=[x_left, x_left + scale*width, y_bottom,␣
 ↪y_bottom + scale*height])
```

```
    ax.scatter(removed_noise['LONGITUDE'], removed_noise['LATITUDE'],␣
↪c=removed_noise['cluster'], alpha = 0.4, s = 10)
    ax.scatter(centroids_user['LONGITUDE'], centroids_user['LATITUDE'],␣
↪c="red", marker='x')
    ax.set_aspect('equal', 'box')
    ax.set_title('Clusters found for user ' + str(group))

    i += 1
```


Clusters found for user 0
Clusters found for user 1
Clusters found for user 2
Clusters found for user 5
Clusters found for user 6
Clusters found for user 7
Clusters found for user 8
Clusters found for user 9
Clusters found for user 10
Clusters found for user 11
Clusters found for user 12
Clusters found for user 14
Clusters found for user 15
Clusters found for user 16
Clusters found for user 17
Clusters found for user 18
1e6
4.8650
4.8649
4.8648
−7700 −7600 −7500 −7400 −7300

Output to file for future retrieval

[30]:
```
dbscan_fit.to_csv('Outputs\\4\\dbscan_each_user.csv', index=False)
```

0.2 Clustering data points of all users

Prepare data using centroids from the first clustering results

[18]:
```
removed_noise = dbscan_fit[dbscan_fit['cluster'] != -1]
removed_noise = removed_noise[removed_noise['cluster'].duplicated(keep=False)]␣
 ↪# remove cluster with only one datapt

mode = lambda x: x.value_counts().index[0]
centroids_user = removed_noise.groupby(['USERID', 'cluster']).agg({'LONGITUDE':␣
 ↪'mean',
                                                                   'LATITUDE':␣
 ↪'mean',
                                                                   'FLOOR':␣
 ↪mode,
                                                                   'BUILDINGID':
 ↪ mode,
                                                                   'SPACEID':␣
 ↪mode,
                                                                  ␣
 ↪'RELATIVEPOSITION': 'count'})
centroids_user = centroids_user.reset_index()
centroids_user.rename(columns={'RELATIVEPOSITION':'pts_count'}, inplace=True)
```

Define hyperparameters

[19]:
```
dist_threshold = 5
min_pts = 2
# Same similarity metric as the first clustering
```

[20]:
```
dbscan = DBSCAN(dist_threshold, min_pts, metric=similarity)
cluster_whole = dbscan.fit_predict(centroids_user)
```

```
D:\USER\Anaconda\envs\geo_env\lib\site-packages\sklearn\utils\validation.py:67:
FutureWarning: Pass min_samples=2 as keyword args. From version 0.25 passing
these as positional arguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

[26]:
```
centroids_user["cluster2"] = cluster_whole
```

[35]:
```
removed_noise = centroids_user[centroids_user['cluster2'] != -1]
removed_noise = removed_noise[removed_noise['cluster2'].duplicated(keep=False)] # remove cluster with only one datapt

mode = lambda x: x.value_counts().index[0]
centroids_whole = removed_noise.groupby(['cluster2']).agg({'USERID': 'count',
                                                           'LONGITUDE': 'mean',
                                                           'LATITUDE': 'mean',
                                                           'FLOOR': mode,
                                                           'BUILDINGID': mode,
                                                           'SPACEID': mode})
centroids_whole = centroids_whole.reset_index()
centroids_whole.rename(columns={'USERID':'pts_count'}, inplace=True)
```

[48]:
```
plt.figure(figsize=(10,6))
plt.axes().set_aspect('equal', 'box')
plt.imshow(floorplan, extent=[x_left, x_left + scale*width, y_bottom, y_bottom + scale*height])
plt.scatter(removed_noise['LONGITUDE'], removed_noise['LATITUDE'], c="grey", s= 10)
plt.scatter(centroids_whole['LONGITUDE'], centroids_whole['LATITUDE'], c=centroids_whole['pts_count'], cmap='autumn_r', marker='x')

cbar = plt.colorbar()
cbar.ax.get_yaxis().set_ticks([i for i in range(2, centroids_whole['pts_count'].max()+1)])
cbar.ax.get_yaxis().labelpad = 15
cbar.ax.set_ylabel('Cluster size', rotation=270)

plt.show()
```

[39]:
```
centroids_user.to_csv('Outputs\\4\\dbscan_all_users.csv', index=False)
centroids_whole.to_csv('Outputs\\4\\dbscan_centroids.csv', index=False)
```

5 cluster_trajectories

October 18, 2020

[20]:
```
import pandas as pd
import numpy as np
import datetime
from pyproj import Transformer
from geopy.distance import distance
import matplotlib.pyplot as plt
from matplotlib import cm
import utils
```

[2]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) { return false; }
```

```
<IPython.core.display.Javascript object>
```

[80]:
```
uji_data = pd.read_csv('Data\\AllData.csv')

# floorplan for visualisation
floorplan = plt.imread("Data\\UJI_B012_floorplan.png")
```

0.1 Prepare data

Retrieve trajectories from data, such that the time threshold between successive points in a trajectory is less than 1 minute, and each trajectory consists at least 10 points.

[4]:
```
all_sequences = []
uji_data['TIME'] = pd.to_datetime(uji_data['TIMESTAMP'], unit='s').dt.time

for user in pd.unique(uji_data['USERID']):
    userdata = uji_data[uji_data['USERID'] == user]
    userdata.sort_values(by='TIMESTAMP', ascending=True)
    userdata = userdata.reset_index(drop=True)

    timeseries = userdata['TIMESTAMP']

    start = 0
    for i in range(userdata.shape[0] - 1):
        if timeseries[i+1] - timeseries[i] > 60: # 1 minute
```

```
            traj = userdata.iloc[start:i+1][['TIME', 'LATITUDE', 'LONGITUDE']]
            sequence = list(zip(traj['TIME'], traj['LATITUDE'],␣
↪traj['LONGITUDE']))

            if len(sequence) > 9:
                all_sequences.append(sequence)

            start = i + 1
```

[5]:
```
sim = np.identity(len(all_sequences))

for i in range(len(all_sequences)):
    for j in range(i+1, len(all_sequences)):
        sim[i,j] = utils.edit_distance_real(all_sequences[i], all_sequences[j])

tril = np.tril_indices_from(sim, -1)
triu = np.triu_indices_from(sim, 1)
sim[tril] = sim[triu]
```

[6]:
```
# print length of trajectories
for sequence in all_sequences:
    print(len(sequence))
```

```
10
11
10
11
18
10
11
11
10
10
10
15
18
21
43
37
11
21
10
526
138
10
```

0.2 Cluster trajectories

[7]:
```
from sklearn.cluster import DBSCAN
```

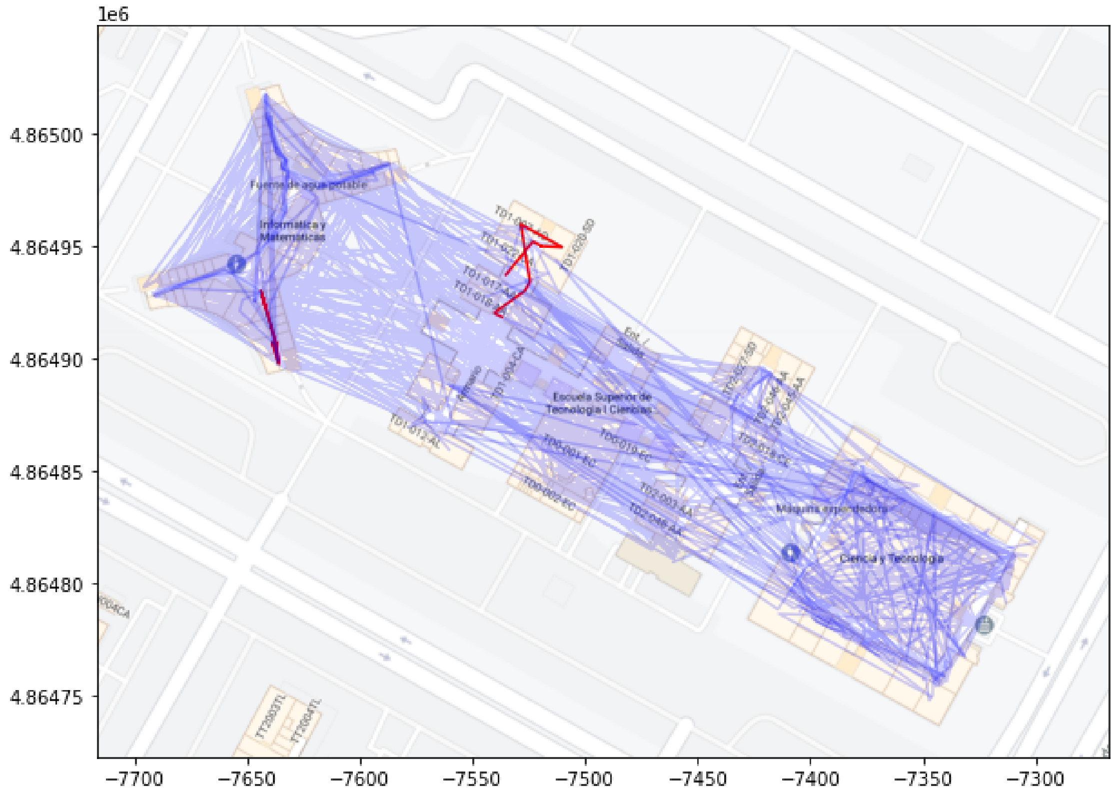
[8]:
```
dbscan = DBSCAN(5, 2, metric='precomputed')
```

```
D:\USER\Anaconda\envs\geo_env\lib\site-packages\sklearn\utils\validation.py:67:
FutureWarning: Pass min_samples=2 as keyword args. From version 0.25 passing
these as positional arguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

[9]:
```
clusters = dbscan.fit_predict(sim)
```

0.3 Visualize trajectories

[82]:
```
x_left, y_bottom = -7717, 4864723
width, height = floorplan.shape[1], floorplan.shape[0]
scale = 0.40

plt.figure(figsize=(10,10))
plt.axes().set_aspect('equal', 'box')
plt.imshow(floorplan, extent=[x_left, x_left + scale*width, y_bottom, y_bottom␣
 ↪+ scale*height])

for i in range(len(all_sequences)):
    sequence = all_sequences[i]
    lat = [x[1] for x in sequence]
    long = [x[2] for x in sequence]
    color = "red" if clusters[i] != -1 else "blue"
    alpha = 0.2 if clusters[i] == -1 else 1
    plt.plot(long, lat, color=color, alpha=alpha)

plt.show()
```



[74]:
```
traj_in_clusters = [[] for _ in range(clusters.max() + 1)]
for i in range(len(all_sequences)):
    if clusters[i] != -1:
        traj_in_clusters[clusters[i]].append(sequence)
```

[75]:
```
traj_in_clusters = [c for c in traj_in_clusters if len(c) > 1]
```

[84]:
```
plt.figure(figsize=(10,10))
cmap = cm.get_cmap('rainbow')

for i in range(len(traj_in_clusters)):
    ax = plt.subplot(1, 1, i+1)
    ax.imshow(floorplan, extent=[x_left, x_left + scale*width, y_bottom, y_bottom + scale*height])
    c = traj_in_clusters[i]

    for j in range(len(c)):
        sequence = c[j]
        rgba = cmap(j/(len(c)-1))
        lat = [x[1] for x in sequence]
        long = [x[2] for x in sequence]
```

```
        ax.plot(long, lat, color=rgba[:3])
        ax.set_aspect('equal', 'box')

plt.show()
```

[79]:
```
# # convert all timestamps to integer
# for c in traj_in_clusters:
#     for t in c:
#         print(t)
#         for i in range(len(t)):
#             time = (t[i][0].hour * 60 + t[i][0].minute) * 60 + t[i][0].second
#             t[i] = (time, t[i][1], t[i][2])

import json

write_file = open('Outputs\\3\\edr_clusters.txt', 'w')
write_file.write(json.dumps(traj_in_clusters))
write_file.close()
```

6 cluster_users

October 18, 2020

[4]:
```
import pandas as pd
import numpy as np
import datetime
import warnings
import matplotlib.pyplot as plt
import seaborn as sns
import utils
warnings.filterwarnings('ignore')
```

[1]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) { return false; }
```

```
<IPython.core.display.Javascript object>
```

[3]:
```
uji_data = pd.read_csv('Data\\AllData.csv')
```

0.1 Preprocess data

[5]:
```
uji_data['DATE'] = pd.to_datetime(uji_data['TIMESTAMP'],unit='s').dt.normalize()
uji_data['TIME'] = pd.to_datetime(uji_data['TIMESTAMP'],unit='s').dt.time
uji_data['DAY'] = uji_data['DATE'].dt.dayofweek
uji_data.iloc[:, -11:]
```

[5]:
```
           LATITUDE  FLOOR  BUILDINGID  SPACEID  RELATIVEPOSITION  USERID  \
0      4.864921e+06      2           1      106                 2       2
1      4.864934e+06      2           1      106                 2       2
2      4.864950e+06      2           1      103                 2       2
3      4.864934e+06      2           1      102                 2       2
4      4.864982e+06      0           0      122                 2      11
…               …     …           …       …                 …     …
20281  4.864796e+06      3           2        0                 0       0
20282  4.864792e+06      3           2        0                 0       0
20283  4.864903e+06      0           0        0                 0       0
20284  4.864905e+06      0           0        0                 0       0
20285  4.864904e+06      0           0        0                 0       0

       PHONEID   TIMESTAMP       DATE       TIME  DAY
```

```
0            23  1371713733 2013-06-20  07:35:33    3
1            23  1371713691 2013-06-20  07:34:51    3
2            23  1371714095 2013-06-20  07:41:35    3
3            23  1371713807 2013-06-20  07:36:47    3
4            13  1369909710 2013-05-30  10:28:30    3
…         …         …          …        …   …
20281        13  1381156711 2013-10-07  14:38:31    0
20282        13  1381156730 2013-10-07  14:38:50    0
20283        13  1381247781 2013-10-08  15:56:21    1
20284        13  1381247807 2013-10-08  15:56:47    1
20285        13  1381247836 2013-10-08  15:57:16    1

[20286 rows x 11 columns]
```

0.2 Feature Extraction

[6]:

```
def average_building_duration(user_df):
    # group by day, find the duration of each day, return the average

    user_df = user_df.sort_values(by='TIMESTAMP', ascending=True)
    user_df = user_df.reset_index()

    buildings = user_df['BUILDINGID']
    days = user_df['DATE']
    time = user_df['TIME']

    ret_df = pd.DataFrame(columns=['BUILDINGID', 'DURATION'])
    init_duration = datetime.timedelta()
    duration_dict = {building: datetime.timedelta() for building in buildings.
↪unique()}
    day_count = 0
    start = 0

    for t in range(user_df.shape[0]):
        if t < user_df.shape[0] - 1:
            assert days[t+1] >= days[t] or time[t+1] >= time[t], datetime.
↪datetime.fromtimestamp(user_df['TIMESTAMP'][t+1]).strftime('%Y-%m-%d %H:%M:
↪%S') + ', ' + days[t+1].strftime('%Y-%m-%d') + ', ' + time[t+1].strftime('%H:
↪%M:%S') + '\n' + datetime.datetime.fromtimestamp(user_df['TIMESTAMP'][t]).
↪strftime('%Y-%m-%d %H:%M:%S') + ', ' + days[t].strftime('%Y-%m-%d') + ', ' +␣
↪time[t].strftime('%H:%M:%S') + '\n'

        elif t == user_df.shape[0] - 1: # last row
            day_count += 1
            duration_dict[buildings[t]] = duration_dict.get(buildings[t],␣
↪init_duration) + utils.time_difference(time[t], time[start])
```

```
            for bldng in duration_dict:
                # calculate average over all visiting days and add to returning␣
 ↪data frame
                ret_df = ret_df.append({'BUILDINGID': bldng, 'DURATION':␣
 ↪duration_dict[bldng] / day_count}, ignore_index=True)

        elif days[t+1] != days[t]: # change day
            day_count += 1
            duration_dict[buildings[t]] = duration_dict.get(buildings[t],␣
 ↪init_duration) + utils.time_difference(time[t], time[start])
            start = t+1

        elif buildings[t+1] != buildings[t]: # change of building
            duration_dict[buildings[t]] = duration_dict.get(buildings[t],␣
 ↪init_duration) + utils.time_difference(time[t], time[start])
            start = t+1

    return ret_df.set_index('BUILDINGID').T
```

[7]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import normalize

def translate_linger_centers_textual(user_df):
    centers = []

    for _, centroid in user_df.iterrows():
        centers += ["B{}F{}S{}".format(int(centroid.BUILDINGID), int(centroid.
 ↪FLOOR), int(centroid.SPACEID))]*int(centroid.pts_count)

    return " ".join(centers)

# perceive cluster centres as words, normalize vectors and use euclidean
# clusters centres of where user tends to linger
def cluster_centers_vectors(user_df):
    corpus = user_df.groupby('USERID').apply(translate_linger_centers_textual)
    vectorizer = TfidfVectorizer()
    X = normalize(vectorizer.fit_transform(corpus))
    X_df = pd.DataFrame.sparse.from_spmatrix(X)
    words = vectorizer.get_feature_names()
    X_df = X_df.rename(lambda x: words[x], axis='columns')
    return X_df.fillna(0)
```

[10]:

```
# unique number of days of visit to campus
f1 = uji_data.groupby('USERID')['DATE'].apply(lambda x: len(x.dt.normalize().
 ↪unique()))
f1 = f1.rename("Days of visit")
```

```python
# average uji_data of visit to campus
f2 = uji_data.groupby(['USERID', 'DATE'])['TIME'].apply(lambda x: utils.
 ↪time_difference(max(x),min(x))).groupby('USERID').apply(lambda x: x.mean())
f2 = f2.rename("Average Duration of visit")

# average duration of staying in a building in different days
f3 = uji_data.groupby('USERID').apply(average_building_duration)
f3.index = [x[0] for x in f3.index.tolist()]
f3.replace({pd.NaT: datetime.timedelta()}, inplace=True)
f3 = f3.rename(lambda x: "Average Duration in Building " + str(x),␣
 ↪axis='columns')

# the median time of day when user visits campus
f4 = uji_data.groupby(['USERID', 'DATE'])['TIME'].apply(utils.datetime_median).
 ↪groupby('USERID').apply(utils.datetime_median)
f4 = f4.rename("Median Time of visit")

# the days when user visits campus
f5 = uji_data.groupby(['USERID','DAY'])['DATE'].apply(lambda x: len(x.dt.
 ↪normalize().unique())).unstack()
f5.replace({np.nan: 0}, inplace=True)
f5 = f5.rename(lambda x: "Visit Count on Day " + str(x), axis='columns')

# clusters centres of where user tends to linger
linger_centers = pd.read_csv('Outputs\\4\\dbscan_all_users.csv')
f6 = cluster_centers_vectors(linger_centers)
```

[11]:
```python
X = pd.concat([f1, f2, f3, f4, f5, f6], axis=1)
X.iloc[:, 1:5] = X.iloc[:, 1:5] / pd.to_timedelta(1, unit='D')
X.iloc[:, 5] = X.iloc[:, 5].apply(lambda a: a.hour + a.minute/60.0)
X.iloc[:, 7:] = X.iloc[:, 7:].fillna(0)
X
```

[11]:
```
    Days of visit  Average Duration of visit  Average Duration in Building 0  \
0               9                   0.108666                        0.322558
1               2                   0.121065                        0.058808
2               1                   0.093264                        0.000000
3               1                   0.006875                        0.000000
4               1                   0.031863                        0.000000
5               1                   0.106782                        0.000000
6               1                   0.083495                        0.000000
7               1                   0.129734                        0.000000
8               1                   0.082743                        0.000000
9               1                   0.108692                        0.000000
10              1                   0.106181                        0.000000
11              4                   0.113501                        0.000000
```

12	1	0.073646	0.000000
13	1	0.085220	0.000000
14	1	0.113206	0.000000
15	1	0.043727	0.000000
16	1	0.078171	0.000000
17	1	0.105359	0.000000
18	1	0.023403	0.000000

	Average Duration in Building 1	Average Duration in Building 2	\
0	0.000000	0.000000	
1	0.000000	0.000000	
2	0.000000	0.093264	
3	0.000000	0.006875	
4	0.031863	0.000000	
5	0.000000	0.106782	
6	0.000000	0.083495	
7	0.000000	0.129734	
8	0.000000	0.082743	
9	0.108692	0.000000	
10	0.000000	0.106181	
11	0.000000	0.012037	
12	0.000000	0.073646	
13	0.000000	0.085220	
14	0.000000	0.113206	
15	0.000000	0.043727	
16	0.000000	0.078171	
17	0.000000	0.105359	
18	0.023403	0.000000	

	Median Time of visit	Visit Count on Day 0	Visit Count on Day 1	\
0	11.066667	1.0	3.0	
1	15.716667	1.0	0.0	
2	8.466667	0.0	0.0	
3	9.266667	0.0	0.0	
4	13.916667	0.0	0.0	
5	9.483333	0.0	0.0	
6	8.016667	0.0	0.0	
7	9.516667	0.0	0.0	
8	9.333333	0.0	0.0	
9	9.433333	0.0	0.0	
10	9.450000	0.0	0.0	
11	15.283333	0.0	1.0	
12	8.100000	0.0	0.0	
13	8.016667	0.0	0.0	
14	9.383333	0.0	0.0	
15	10.150000	0.0	0.0	
16	8.416667	0.0	0.0	

```
17             9.550000                     0.0                    0.0
18             6.533333                     0.0                    0.0

    Visit Count on Day 2  Visit Count on Day 3  …  b2f3s214  b2f3s215  \
0                    0.0                   3.0  …  0.000000  0.000000
1                    1.0                   0.0  …  0.000000  0.000000
2                    0.0                   1.0  …  0.285714  0.285714
3                    0.0                   1.0  …  0.000000  0.000000
4                    0.0                   1.0  …  0.000000  0.000000
5                    0.0                   1.0  …  0.000000  0.000000
6                    0.0                   1.0  …  0.000000  0.000000
7                    0.0                   1.0  …  0.000000  0.000000
8                    0.0                   1.0  …  0.000000  0.000000
9                    0.0                   1.0  …  0.000000  0.000000
10                   0.0                   1.0  …  0.000000  0.000000
11                   0.0                   2.0  …  0.000000  0.000000
12                   0.0                   1.0  …  0.000000  0.000000
13                   0.0                   1.0  …  0.000000  0.000000
14                   0.0                   1.0  …  0.000000  0.000000
15                   0.0                   1.0  …  0.000000  0.000000
16                   0.0                   1.0  …  0.000000  0.000000
17                   0.0                   1.0  …  0.000000  0.000000
18                   0.0                   1.0  …  0.000000  0.000000

    b2f3s216  b2f3s223  b2f3s230  b2f3s236  b2f3s241  b2f3s247  b2f4s129  \
0   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
2   0.285714  0.285714  0.285714  0.285714  0.285714  0.285714  0.000000
3   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
4   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.707107
5   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
6   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
7   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
8   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
9   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
10  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
11  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
12  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
13  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
14  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
15  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
16  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
17  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
18  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000

    b2f4s141
0   0.000000
```

```
1    0.000000
2    0.000000
3    0.000000
4    0.707107
5    0.000000
6    0.000000
7    0.000000
8    0.000000
9    0.000000
10   0.000000
11   0.000000
12   0.000000
13   0.000000
14   0.000000
15   0.000000
16   0.000000
17   0.000000
18   0.000000

[19 rows x 159 columns]
```

[12]:
```
from sklearn import preprocessing

# Normalize values for training
# - prevent exessive effect of features with smaller variance
# Normalize > standardization in the sense that it does not make any
# assumptions abt the distribution of the features.
# Even of gaussian, small dataset means inaccuracy or bias in estimating mean
↪and s.d.
scaler = preprocessing.MinMaxScaler()
scaled_X = X.copy()
# only from f1 to f4, since f5 is binned and f6 is already normalized
scaled_X.iloc[:, 0:11] = scaler.fit_transform(scaled_X.iloc[:, 0:11])
scaled_X
```

[12]:
```
    Days of visit  Average Duration of visit  Average Duration in Building 0  \
0           1.000                   0.828524                        1.000000
1           0.125                   0.929439                        0.182317
2           0.000                   0.703156                        0.000000
3           0.000                   0.000000                        0.000000
4           0.000                   0.203391                        0.000000
5           0.000                   0.813189                        0.000000
6           0.000                   0.623646                        0.000000
7           0.000                   1.000000                        0.000000
8           0.000                   0.617522                        0.000000
9           0.000                   0.828733                        0.000000
10          0.000                   0.808290                        0.000000
```

11	0.375	0.867876	0.000000
12	0.000	0.543476	0.000000
13	0.000	0.637683	0.000000
14	0.000	0.865473	0.000000
15	0.000	0.299953	0.000000
16	0.000	0.580311	0.000000
17	0.000	0.801602	0.000000
18	0.000	0.134527	0.000000

	Average Duration in Building 1	Average Duration in Building 2 \
0	0.000000	0.000000
1	0.000000	0.000000
2	0.000000	0.718887
3	0.000000	0.052993
4	0.293153	0.000000
5	0.000000	0.823089
6	0.000000	0.643590
7	0.000000	1.000000
8	0.000000	0.637791
9	1.000000	0.000000
10	0.000000	0.818449
11	0.000000	0.092783
12	0.000000	0.567669
13	0.000000	0.656883
14	0.000000	0.872602
15	0.000000	0.337051
16	0.000000	0.602552
17	0.000000	0.812115
18	0.215313	0.000000

	Median Time of visit	Visit Count on Day 0	Visit Count on Day 1 \
0	0.493648	1.0	1.000000
1	1.000000	1.0	0.000000
2	0.210526	0.0	0.000000
3	0.297641	0.0	0.000000
4	0.803993	0.0	0.000000
5	0.321234	0.0	0.000000
6	0.161525	0.0	0.000000
7	0.324864	0.0	0.000000
8	0.304900	0.0	0.000000
9	0.315789	0.0	0.000000
10	0.317604	0.0	0.000000
11	0.952813	0.0	0.333333
12	0.170599	0.0	0.000000
13	0.161525	0.0	0.000000
14	0.310345	0.0	0.000000
15	0.393829	0.0	0.000000

16	0.205082	0.0	0.000000
17	0.328494	0.0	0.000000
18	0.000000	0.0	0.000000

	Visit Count on Day 2	Visit Count on Day 3	…	b2f3s214	b2f3s215	\
0	0.0	1.000000	…	0.000000	0.000000	
1	1.0	0.000000	…	0.000000	0.000000	
2	0.0	0.333333	…	0.285714	0.285714	
3	0.0	0.333333	…	0.000000	0.000000	
4	0.0	0.333333	…	0.000000	0.000000	
5	0.0	0.333333	…	0.000000	0.000000	
6	0.0	0.333333	…	0.000000	0.000000	
7	0.0	0.333333	…	0.000000	0.000000	
8	0.0	0.333333	…	0.000000	0.000000	
9	0.0	0.333333	…	0.000000	0.000000	
10	0.0	0.333333	…	0.000000	0.000000	
11	0.0	0.666667	…	0.000000	0.000000	
12	0.0	0.333333	…	0.000000	0.000000	
13	0.0	0.333333	…	0.000000	0.000000	
14	0.0	0.333333	…	0.000000	0.000000	
15	0.0	0.333333	…	0.000000	0.000000	
16	0.0	0.333333	…	0.000000	0.000000	
17	0.0	0.333333	…	0.000000	0.000000	
18	0.0	0.333333	…	0.000000	0.000000	

	b2f3s216	b2f3s223	b2f3s230	b2f3s236	b2f3s241	b2f3s247	b2f4s129	\
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
2	0.285714	0.285714	0.285714	0.285714	0.285714	0.285714	0.000000	
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.707107	
5	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
6	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
7	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
8	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
10	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
11	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
12	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
13	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
14	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
15	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
16	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
17	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
18	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	

b2f4s141

```
0    0.000000
1    0.000000
2    0.000000
3    0.000000
4    0.707107
5    0.000000
6    0.000000
7    0.000000
8    0.000000
9    0.000000
10   0.000000
11   0.000000
12   0.000000
13   0.000000
14   0.000000
15   0.000000
16   0.000000
17   0.000000
18   0.000000

[19 rows x 159 columns]
```

0.3 Clustering

[13]:
```
from sklearn.cluster import KMeans, AgglomerativeClustering
from BisectingKMeans import BisectingKMeans
```

[14]:
```
from sklearn.metrics import silhouette_score

clustering_algo = ["K-Means", "Bisecting K-Means", "Agglomerative"]
agglo_linkages = ["ward", "complete", "average", "single"]
models = [KMeans(), BisectingKMeans(), AgglomerativeClustering()]
evaluation_df = pd.DataFrame(columns=['Algorithm', 'n',␣
↪'Agglomerative_Linkage', 'Inertia', 'Sillhoette'])
clustering_results = pd.DataFrame(columns=[i for i in range(X.shape[0])])

def compute_inertia(agglo_model, X):
    assert agglo_model.labels_ is not None
    centroids = compute_centres(X, agglo_model.labels_, agglo_model.n_clusters_)
    inertia = 0

    for l in range(agglo_model.n_clusters_):
        centroid = centroids[l, :]
        label_x = X.iloc[np.where(agglo_model.labels_ == l)]
        inertia += np.sum(np.sum(np.square(label_x - centroid)))
```

```
    return inertia

def compute_centres(X, labels, n_clusters):
    assert labels is not None
    centroids = np.zeros((n_clusters,X.shape[1]))

    for l in range(n_clusters):
        label_x = X.iloc[np.where(labels == l)]
        centroid = np.mean(label_x, 0)
        centroids[l,:] = centroid

    return centroids

def fit_eval_to_dict(model, data, is_agglo=False):
    eval_dict = dict()
    model.fit(data)
    if is_agglo:
        eval_dict['Inertia'] = compute_inertia(model, data)
    else:
        eval_dict['Inertia'] = model.inertia_
    eval_dict['Sillhoette'] = silhouette_score(data, model.labels_)
    return eval_dict

for n in range(2, 10):
    for i in range(len(clustering_algo)):
        model = models[i]
        model.set_params(**{'n_clusters': n})
        if i == len(clustering_algo) - 1: # is agglomerative clustering
            for j in range(len(agglo_linkages)):
                print("Fitting using {} with n={} & linkage={}".
↪format(clustering_algo[i], n, agglo_linkages[j]))
                model.set_params(**{"linkage": agglo_linkages[j]})
                eval_dict = fit_eval_to_dict(model, scaled_X, is_agglo=True)
                eval_dict['Agglomerative_Linkage'] = agglo_linkages[j]
                eval_dict['Algorithm'] = clustering_algo[i]
                eval_dict['n'] = n
                evaluation_df = evaluation_df.append(eval_dict,␣
↪ignore_index=True)
                clustering_results = clustering_results.append(pd.Series(model.
↪labels_), ignore_index=True)

                # use results of agglomerative for kmeans
                centroids = compute_centres(scaled_X, model.labels_, model.
↪n_clusters_)
                kmeans_model = KMeans(n_clusters=n, init=centroids)
                eval_dict = fit_eval_to_dict(kmeans_model, scaled_X)
                eval_dict['Agglomerative_Linkage'] = agglo_linkages[j]
```

```
                eval_dict['Algorithm'] = clustering_algo[i] + " then K-Means"
                eval_dict['n'] = n
                evaluation_df = evaluation_df.append(eval_dict,␣
↪ignore_index=True)
                clustering_results = clustering_results.append(pd.Series(model.
↪labels_), ignore_index=True)
        else:
            print("Fitting using {} with n={}".format(clustering_algo[i], n))
            eval_dict = fit_eval_to_dict(model, scaled_X)
            eval_dict['Algorithm'] = clustering_algo[i]
            eval_dict['n'] = n
            evaluation_df = evaluation_df.append(eval_dict, ignore_index=True)
            clustering_results = clustering_results.append(pd.Series(model.
↪labels_), ignore_index=True)
```

```
Fitting using K-Means with n=2
Fitting using Bisecting K-Means with n=2
Fitting using Agglomerative with n=2 & linkage=ward
Fitting using Agglomerative with n=2 & linkage=complete
Fitting using Agglomerative with n=2 & linkage=average
Fitting using Agglomerative with n=2 & linkage=single
Fitting using K-Means with n=3
Fitting using Bisecting K-Means with n=3
Fitting using Agglomerative with n=3 & linkage=ward
Fitting using Agglomerative with n=3 & linkage=complete
Fitting using Agglomerative with n=3 & linkage=average
Fitting using Agglomerative with n=3 & linkage=single
Fitting using K-Means with n=4
Fitting using Bisecting K-Means with n=4
Fitting using Agglomerative with n=4 & linkage=ward
Fitting using Agglomerative with n=4 & linkage=complete
Fitting using Agglomerative with n=4 & linkage=average
Fitting using Agglomerative with n=4 & linkage=single
Fitting using K-Means with n=5
Fitting using Bisecting K-Means with n=5
Fitting using Agglomerative with n=5 & linkage=ward
Fitting using Agglomerative with n=5 & linkage=complete
Fitting using Agglomerative with n=5 & linkage=average
Fitting using Agglomerative with n=5 & linkage=single
Fitting using K-Means with n=6
Fitting using Bisecting K-Means with n=6
Fitting using Agglomerative with n=6 & linkage=ward
Fitting using Agglomerative with n=6 & linkage=complete
Fitting using Agglomerative with n=6 & linkage=average
Fitting using Agglomerative with n=6 & linkage=single
Fitting using K-Means with n=7
Fitting using Bisecting K-Means with n=7
```

```
Fitting using Agglomerative with n=7 & linkage=ward
Fitting using Agglomerative with n=7 & linkage=complete
Fitting using Agglomerative with n=7 & linkage=average
Fitting using Agglomerative with n=7 & linkage=single
Fitting using K-Means with n=8
Fitting using Bisecting K-Means with n=8
Fitting using Agglomerative with n=8 & linkage=ward
Fitting using Agglomerative with n=8 & linkage=complete
Fitting using Agglomerative with n=8 & linkage=average
Fitting using Agglomerative with n=8 & linkage=single
Fitting using K-Means with n=9
Fitting using Bisecting K-Means with n=9
Fitting using Agglomerative with n=9 & linkage=ward
Fitting using Agglomerative with n=9 & linkage=complete
Fitting using Agglomerative with n=9 & linkage=average
Fitting using Agglomerative with n=9 & linkage=single
```

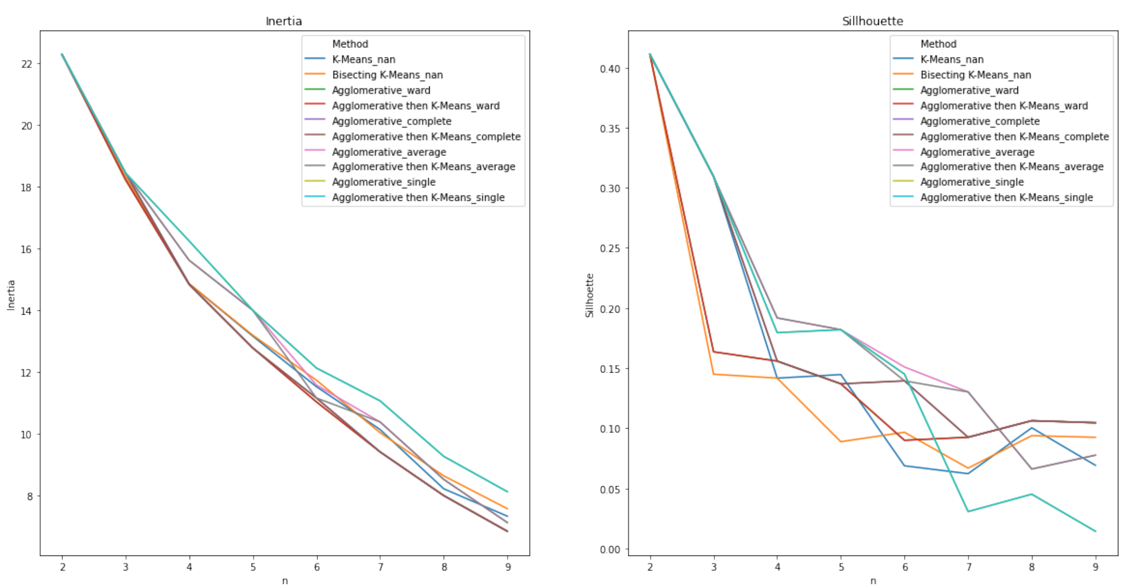
0.4 Evaluation

[15]:

```python
plt.figure(figsize=(20,10))
eval_df_to_plot = evaluation_df.copy()
eval_df_to_plot['Method'] = evaluation_df['Algorithm'] + '_' +␣
↪evaluation_df['Agglomerative_Linkage'].astype(str)
ax = plt.subplot(1, 2, 1)
ax.set_title("Inertia")
sns.lineplot(data=eval_df_to_plot, x="n", y="Inertia", hue="Method")
ax = plt.subplot(1, 2, 2)
ax.set_title("Sillhouette")
sns.lineplot(data=eval_df_to_plot, x="n", y="Sillhoette", hue="Method")
```

[15]:

```
<AxesSubplot:title={'center':'Sillhouette'}, xlabel='n', ylabel='Sillhoette'>
```



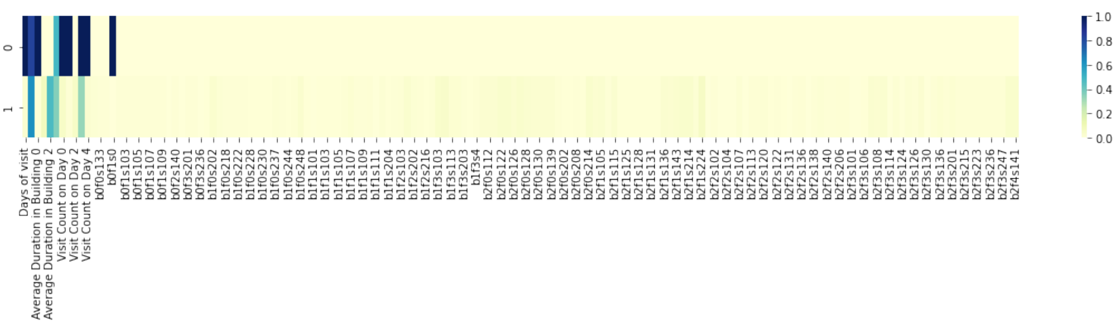
[16]:
```python
clustering_results[evaluation_df['n'] == 2]
```

[16]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[17]:
```python
centroids = compute_centres(scaled_X, clustering_results.iloc[0,:], 2)
centroids = pd.DataFrame(centroids)
centroids.columns = X.columns

plt.figure(figsize=(20,2))
sns.heatmap(centroids, cmap="YlGnBu")
```

[17]: <AxesSubplot:>



0.5 Using network to extract more probable clusters

[18]:
```python
from itertools import combinations
cluster_network = np.zeros(shape=(scaled_X.shape[0], scaled_X.shape[0]))

for (_, data) in clustering_results.iterrows():
    count = 0
    i = 0
    while count != scaled_X.shape[0]:
```

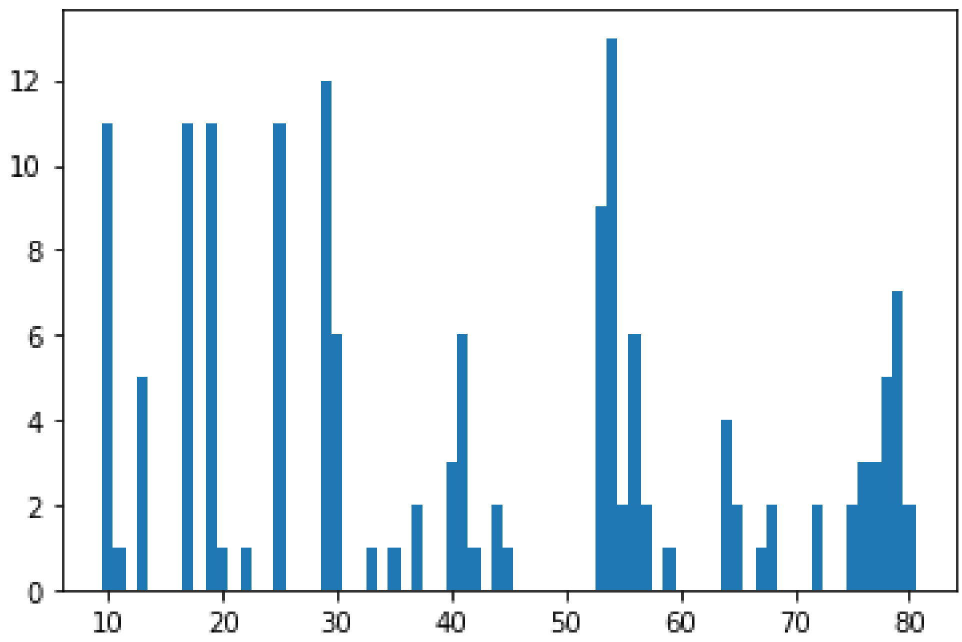
```
        cluster_pts = data[data == i].index
        for pair in list(combinations(cluster_pts, 2)):
            cluster_network[pair[0], pair[1]] += 1
            cluster_network[pair[1], pair[0]] += 1
        i += 1
        count += cluster_pts.size
```

[20]:
```
np.savetxt('Outputs\\6\\cluster_adj_mat.csv', cluster_network, delimiter=',')
```

[21]:
```
cluster_network = pd.DataFrame(cluster_network)
```

[24]:
```
import matplotlib.pyplot as plt
import networkx as nx

plt.figure(figsize=(10,5))
gr = nx.from_pandas_adjacency(cluster_network)
scaled_weights = [gr[u][v]['weight']/10 for u, v in gr.edges()]
nx.draw_networkx(gr, node_size = 500, with_labels=True, width=scaled_weights)
plt.show()
```

[25]:
```
# distribution of weights
weights = [gr[u][v]['weight'] for u, v in gr.edges()]
weights = np.array(weights)
d = np.diff(np.unique(weights)).min()
left = weights.min() - float(d)/2
right = weights.max() + float(d)/2
plt.hist(weights, np.arange(left, right+d, d))
plt.show()
```



[26]:

```
# remove edges when weight less than 70, i.e. remove connections of users if␣
↪they are clustered in the same group
# for less than 70 times
mini_gr = gr.copy()
for u, v, w in gr.edges.data('weight'):
    if w < 70:
        mini_gr.remove_edge(u, v)

for c in nx.connected_components(mini_gr):
    print(c)
```

```
{0}
{1}
{2, 5, 6, 12, 13, 16, 17}
{3}
{4}
{8, 14, 7}
{9}
{10}
{11}
{15}
{18}
```

[27]:

```
cluster_no = 0
use_label = np.zeros(shape=(X.shape[0],), dtype=int)
```

```
for c in nx.connected_components(mini_gr):
    for user in c:
        use_label[user] = cluster_no
    cluster_no += 1

use_label
```

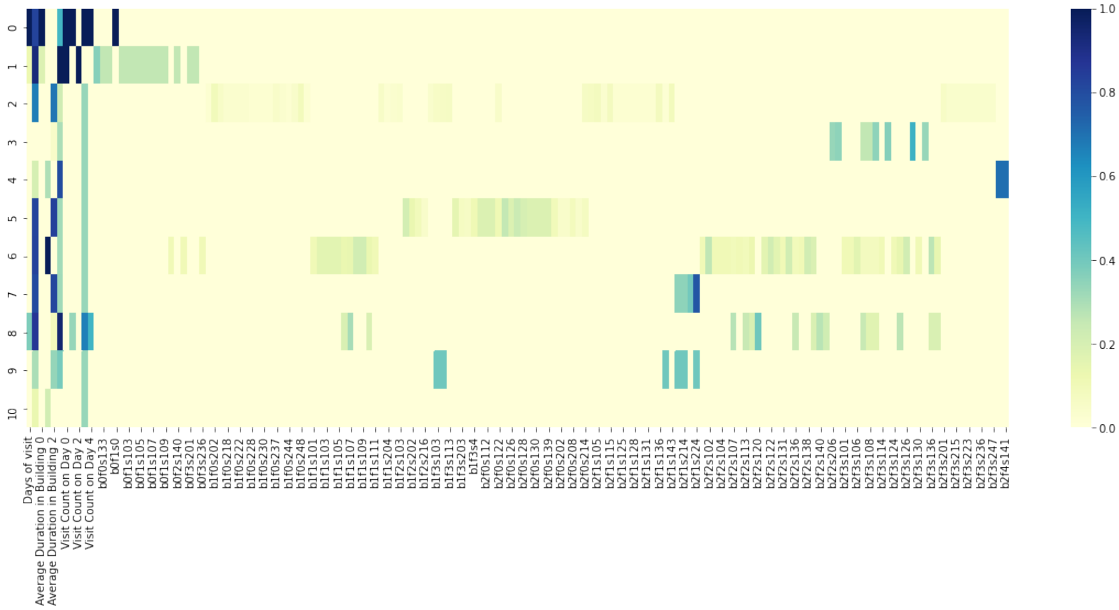
[27]:
```
array([ 0,  1,  2,  3,  4,  2,  2,  5,  5,  6,  7,  8,  2,  2,  5,  9,  2,
        2, 10])
```

[28]:
```
# remove nodes which is not in any cluster
mini_mini_gr = mini_gr.copy()
for c in nx.connected_components(mini_gr):
    if len(c) == 1:
        for user in c:
            mini_mini_gr.remove_node(user)

plt.figure(figsize=(15,5))
scaled_weights = [mini_mini_gr[u][v]['weight']/10 for u, v in mini_mini_gr.
 ↪edges()]
nx.draw_networkx(mini_mini_gr, node_size = 500, with_labels=True,␣
 ↪width=scaled_weights)
plt.show()
```

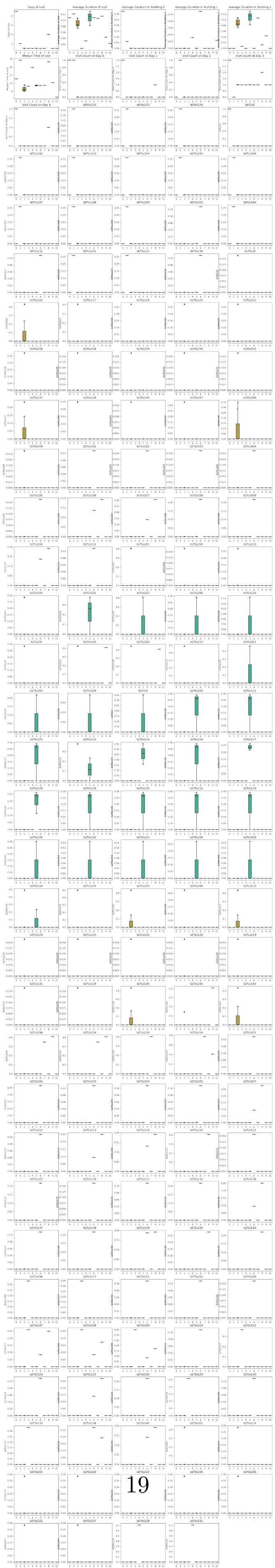
0.6 Cluster Analysis

[32]:
```
centroids = compute_centres(scaled_X, use_label, len([c for c in nx.
 ↪connected_components(mini_gr)]))
centroids = pd.DataFrame(centroids)
centroids.columns = X.columns

plt.figure(figsize=(20,7))
sns.heatmap(centroids, cmap="YlGnBu")
```

[32]:
```
<AxesSubplot:>
```



[29]:
```
# visualize distribution of variables based on cluster from mini_gr
plt.figure(figsize=(20,124), dpi=80)
i = 1
for name, data in X.iteritems(): # use original data to see distribution, not␣
↪scaled
    ax = plt.subplot(32, 5, i)
    sns.boxplot(x=use_label, y=data)
    sns.stripplot(x=use_label, y=data, color='black', size=3, jitter=1)
    ax.set_title(name)
    i += 1
```


Days of visit
Average Duration of visit
Average Duration in Building 0
Average Duration in Building 1
Average Duration in Building 2
Median Time of visit
Visit Count on Day 0
Visit Count on Day 1
Visit Count on Day 2
Visit Count on Day 3
Visit Count on Day 4
b0f0s120
b0f0s133
b0f0s134
b0f1s0

7 Areas of Interest + Optimise User Schedule

October 18, 2020

[1]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) { return false; }
```

```
<IPython.core.display.Javascript object>
```

0.1 Visualisation of Areas of Interest

[15]:
```
import pandas as pd
import matplotlib.pyplot as plt

uji_data = pd.read_csv('Data\\AllData.csv')
dbscan_centroids = pd.read_csv('Outputs\\4\\dbscan_centroids.csv')
edr_trajectories = json.loads(open('Outputs\\3\\edr_clusters.txt').read())

# floorplan for visualisation
floorplan = plt.imread("Data\\UJI_B012_floorplan.png")
```

[30]:
```
from matplotlib import cm

x_left, y_bottom = -7717, 4864723
width, height = floorplan.shape[1], floorplan.shape[0]
scale = 0.40

plt.figure(figsize=(15,10))
plt.axes().set_aspect('equal', 'box')
plt.imshow(floorplan, extent=[x_left, x_left + scale*width, y_bottom, y_bottom␣
 ↪+ scale*height])
plt.scatter(dbscan_centroids['LONGITUDE'], dbscan_centroids['LATITUDE'],␣
 ↪c=dbscan_centroids['pts_count'], cmap='autumn_r', marker='x')

cmap = cm.get_cmap('autumn_r')
for i in range(len(edr_trajectories)):
    c = edr_trajectories[i]
    rgba = cmap(len(c)/dbscan_centroids['pts_count'].max())

    for j in range(len(c)):
        sequence = c[j]
```

```
        lat = [x[1] for x in sequence]
        long = [x[2] for x in sequence]
        plt.plot(long, lat, color=rgba[:3])

# Color bar legend
cbar = plt.colorbar()
cbar.ax.get_yaxis().set_ticks([i for i in range(2,␣
 ↪dbscan_centroids['pts_count'].max()+1)])
cbar.ax.get_yaxis().labelpad = 15
cbar.ax.set_ylabel('Cluster size', rotation=270)

plt.show()
```

0.2 Optimise the Scheduling of Users Access to Site

[2]:
```
from ortools.sat.python import cp_model
from itertools import combinations
import math
from sklearn import preprocessing
```

```python
def solve_day_scheduling(users_count, num_days, cmc_adj_mat, cluster_adj_mat):
    # Normalize cmc_adj_mat and cluster_adj_mat
    scaler = preprocessing.MinMaxScaler()
    cmc_adj_mat = scaler.fit_transform(cmc_adj_mat)
    cluster_adj_mat = scaler.fit_transform(cluster_adj_mat)
    cmc_adj_mat = np.floor(cmc_adj_mat * 1000).astype(int)
    cluster_adj_mat = np.floor(cluster_adj_mat * 1000).astype(int)

    model = cp_model.CpModel()

    allowed = {}
    for u in range(users_count):
        for d in range(num_days):
            allowed[u, d] = model.NewBoolVar('allowed_user%ion%i' % (u, d))

    # only allow 70% of all users each day
    for d in range(num_days):
        model.Add(sum(allowed[u, d] for u in range(users_count)) <= math.floor(users_count * 0.7))

    # allow each user for 3 days in a week
    for u in range(users_count):
        model.Add(sum(allowed[u, d] for d in range(num_days)) == 3)

    max_bool_vars = []
    max_bool_coeffs = []
    min_bool_vars = []
    min_bool_coeffs = []

    for d in range(num_days):
        for pair in list(combinations([x for x in range(users_count)], 2)):
            # if both 1 then award with edge weight in cmc_adj_mat
            # to encourage closely connected users to be allowed on same day
            same_day_var = model.NewBoolVar('%i and %i same on day %i' % (pair[0], pair[1], d))
            model.AddBoolOr([allowed[pair[0], d].Not(),
                             allowed[pair[1], d].Not(),
                             same_day_var])
            max_bool_vars.append(same_day_var)
            max_bool_coeffs.append(cmc_adj_mat[pair[0], pair[1]])

            # if diff day then penalize with edge weight in cluster_adj_mat
            # to discourage same behaviour users to be allowed on same day
            same_not_day_var = model.NewBoolVar('%i and %i not on day %i' % (pair[0], pair[1], d))
            model.AddBoolOr([allowed[pair[0], d],
                             allowed[pair[1], d],
```

```
                                  same_not_day_var])
            diff_day_var = model.NewBoolVar('%i and %i diff on day %i' %␣
↪(pair[0], pair[1], d))
            model.AddBoolOr([same_day_var,
                             same_not_day_var,
                             diff_day_var])
            min_bool_vars.append(diff_day_var)
            min_bool_coeffs.append(cluster_adj_mat[pair[0], pair[1]])

    model.Maximize(
        sum(max_bool_vars[i] * max_bool_coeffs[i] for i in␣
↪range(len(max_bool_vars))) -
        sum(min_bool_vars[i] * min_bool_coeffs[i] for i in␣
↪range(len(min_bool_vars)))
    )

    solver = cp_model.CpSolver()
    solution_printer = cp_model.ObjectiveSolutionPrinter()
    status = solver.SolveWithSolutionCallback(model, solution_printer)

    if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
        if status == cp_model.OPTIMAL:
            print("Optimal is found.")
        else:
            print("Optimal not found but feasible.")

        to_print = ''
        for d in range(num_days):
            to_print += 'Day ' + str(d) + '\n'
            for u in range(users_count):
                if solver.BooleanValue(allowed[u, d]):
                    to_print += '\tUser ' + str(u) + '\n'

        print(to_print)
```

[31]:
```
import numpy as np

cmc_adj_mat = np.loadtxt('Outputs\\3\\cmc_adj_mat.csv', delimiter=',')
cluster_adj_mat = np.loadtxt('Outputs\\6\\cluster_adj_mat.csv', delimiter=',')
```

[32]:
```
solve_day_scheduling(cmc_adj_mat.shape[0], 5, cmc_adj_mat, cluster_adj_mat)
```

```
Solution 0, time = 0.33 s, objective = 21440
Optimal is found.
Day 0
        User 4
```

```
        User 7
        User 8
        User 9
        User 15
Day 1
        User 0
        User 5
        User 6
        User 7
        User 9
        User 10
        User 11
        User 12
        User 13
        User 14
        User 15
        User 16
        User 18
Day 2
        User 0
        User 1
        User 2
        User 3
        User 4
        User 8
        User 10
        User 11
        User 12
        User 13
        User 16
        User 17
        User 18
Day 3
        User 0
        User 1
        User 2
        User 3
        User 4
        User 5
        User 6
        User 7
        User 8
        User 11
        User 14
        User 16
        User 17
Day 4
        User 1
```

```
User 2
User 3
User 5
User 6
User 9
User 10
User 12
User 13
User 14
User 15
User 17
User 18
```