Data Warehousing and Data Mining - CO461

National Institute of Technology Karnataka Surathkal

Computer Science and Engineering

Topic: Geo-social Clustering of Places from Check-in Data

Submitted By: Rahul Kumar (171CO133)

Mudavath Mohit Chauhan (171CO224)

Abstract

Developing an algorithm to cluster places not only based on their location but also their semantics. In particular, two places are considered similar if they are spatially close and visited by people from similar communities. From explosive growth in the availability of location tracking technologies, it became easy to track user locations and movements through custom check-ins. These checks give an idea of the structure of the community of people visiting a place that attracted and integrated into the proposed geosocial cluster, a framework called GeoScop. So far, community discovery is usually made on social networks. In our problem, we are missing any network data. Rather, two people belong to one community if they visit similar geosocial clusters. We are doing this chicken and egg problem with an iterative procedure expectation-maximization and DBSCAN. Extensive experiments on real registration data demonstrate that GeoScop is a semantically significant cluster that cannot be found using existing clustering methods. Also, GeoScop works Six times cleaner in social quality than in modern artistic technique.

Contents

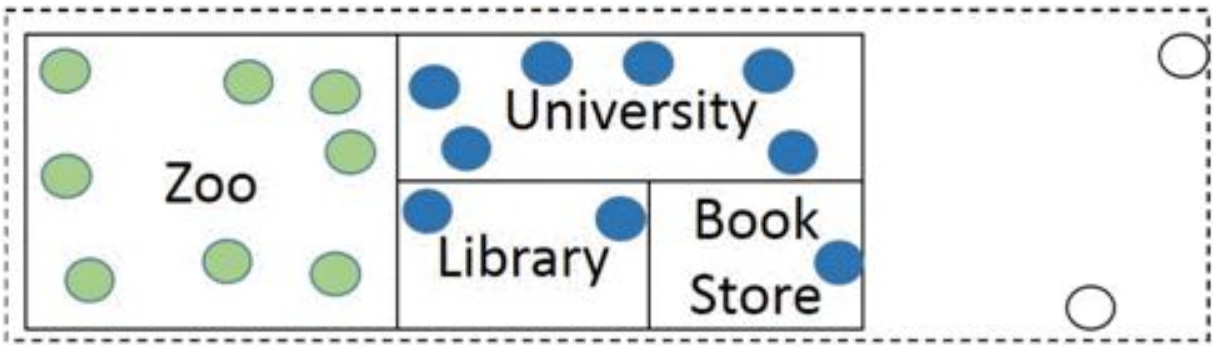
Introduction

Clustering is one of the most common EDA(Exploratory data analysis) techniques used to get an intuition about the structure of the data. It can be defined as identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar. In contrast, data points in different sets are different. This means we try to find homogeneous subgroups within the data such that data points in each group are as similar as possible according to a similarity measure such as euclidean-based distance.

Clustering is used in market segmentation. We find customers that are exactly similar to each other in terms of behaviors, attributes, image segmentation/compression, where we try to group similar regions, document clustering based on topics, etc. Spatial clustering is an unsupervised algorithm to group places into clusters, such that those within a cluster are similar, and places across clusters are dissimilar. Traditionally, the distance function between two locations quantifies the spatial distance between them and ignores semantic similarity.

In specific applications, such a distance function leads to spurious clusters. The paper implementation develops a technique called GeoScop (GEO-Social Clustering of Places) to mine geo-social clusters from check-in data. Due to the ubiquity of GPS-enabled phones, there has been a spurt of services built on check-in data. These check-ins provide a rich form of geographical data, which offers new and exciting insights, compared to raw spatial data.

It is proposed to extend traditional density-based clustering for spatial locations to consider their relationship to a community of people who visit them and when they were visited. In specific, we think the places are based on the Geo-Social data application, which allows users to capture their geographic locations and form communities of users that visit a particular spatial cluster at a given time frame by an operation called check-in.



According to the figure, Let the blue check-ins represent students' visits, and the green check-ins represent tourists. If the check-ins are clustered using traditional spatial-clustering algorithms, all check-ins in the zoo, university, book store, and library will form a single cluster. The two white check-ins would be outliers. However, it is easy to see that there are two geo-social clusters. While students mostly visit the university, library, and book store, the zoo is frequented by tourists, representing a different profile of user communities.

This approach extends DBSCAN by replacing the Euclidean distance threshold for the extents of dense regions by a threshold ε, which considers both the spatial and social distances between places. For two areas, p i and p j , the spatial distance is regarded as the Euclidean distance between p i and p j . In contrast, the social distance should consider the social relationships between the two sets of users, U pi and U pi, checked in p i and p j , respectively. We define and use such a social distance measure, based on the intuition that two places are socially similar.

Background

Conventional spatial clustering is done using the DBSCAN approach. Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm given by Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu in 1996. It is a density-based clustering non-parametric algorithm: which means, given a set of points in some places, it groups together points that are closely packed together, marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away).

The closest work to our problem is DCPGS. Although the formulation of DCPGS is different, it has the same goal of grouping places based on spatial distance and social distance. However, DCPGS assumes the presence of an observable social network to mine communities of users. In our problem, we have access to only the check-in data and lack users' social structure. While it is always useful to have additional information in a social network, this assumption does not seem to be realistic. It is well known that only a few companies, such as Facebook and Twitter, have access to large scale social networks.

For most companies like Yelp, Zomato, TripAdvisor, and credit card agencies, the only source of data is the check-ins. Social network owners are known to not share their data due to competitive business advantage, its inestimable value, and privacy concerns. Therefore, the question arises: How do we perform geo-social clustering without

knowledge of the explicit network among users? Essentially, we face a chicken-and-egg problem. Two users are part of the same community if they visit similar geo-social clusters.

On the other hand, two places are identical if people see them in similar neighborhoods. We solve this problem through the iterative procedure of community detection and geo-social clustering as outlined further. To summarize, our work are as follows:

- We formulate the novel problem of geo-social clustering from check-in data. As a by-product of our problem, we also mine geo-social communities.
- We develop a technique called GeoScop to mine geo-social clusters from check-in data without relying on a social network.
- Extensive experiments on real check-in data establish that GeoScop can mine semantically meaningful clusters and up to 6 times purer than the state-of-the-art techniques.

Improvement

1. Alternative Computation of Social Distance

The authors of the paper have proposed implementing a concept of Social distance to do the process of community initialization. Social Distance is a calculation of the closeness or farness of two U pi and U pj that may belong to the same or different community. Depending on the Threshold set for determining a social cluster or a user community cluster, the social distance conventionally determines whether two users U pi and U pj who visit places pi and pj within a specific interval of time will form a community such that it becomes a social cluster or user community cluster.

Social Distance. Given, the community profiles of places P1 and P2, the social distance sdist(P1, P2) is defined as the following.

$$sdist(p_1, p_2) = 1 - \frac{\sum_{\forall C_i \in \mathbb{C}} \min\{pr(p_1)[i], pr(p_2)[i]\}}{\sum_{\forall C_i \in \mathbb{C}} \max\{pr(p_1)[i], pr(p_2)[i]\}}$$

It was found that this step is rather redundant in nature and that the author may have done it for modularity purposes.

Instead of this implementation, it was decided to split the social distance calculation and merge it with the previous and next steps. This would mean that during the last stage of each user's mining cluster profile, the closeness of a user concerning other users present is also calculated. During the process of community initialization, the proximity measure of each user cluster profile is first normalized and calculated.

This step directly computes the social distance between places pi and pj depending on the proximity of users visiting pi and pj based on the check-ins done by them in some time intervals. This would reduce the number of steps and reduce the number of computations needed in the algorithm. In turn, these measures increase the efficiency of the algorithm and reduce the latency or delay encountered while performing community mining.

2. The different metric was chosen for user similarity calculation

The geo-clusters are formed from the user check-ins dataset where user data is not used for the clustering. The next step number of visits made by each user for the obtained clusters in the above step is calculated, and user similarity is found by the Gaussian mixtures metric. With this community-Initialization is made, and the next community profile is obtained.

Once community clusters are formed, community-profile is calculated which number of visits made by a user cluster to a particular place.

Algorithm

Step 0: Determining the initial input parameters into the model

- The first step is to identify the input parameters

Step 1: We Initially perform DBSCAN on the dataset of geographical

- maxD - spatial radius of a cluster for DBSCAN, to be specified in meters Values: 0<maxD<1000 Default Value: 120 meters
- dataset to be used: Gowalla

Step 2: Locations to cluster them based on euclidean distance.

For the initial set of clusters, we simply perform DBSCAN without considering the social aspect

- Divide the dataset into n dimensions
- For each point in the dataset, use DBSCAN to form an ‘n’ dimensional shape, and then count how many data points fall within that shape (cluster).
- Through DBSCAN iteratively expand the cluster, by going through each individual point within the cluster, and counting the number of other data points nearby.

Step 3: Mining cluster profile of users based on check-in data.

- The check-in data is taken into consideration.
- For each user, the number of visits done to a particular Location is analyzed.
- The check-ins done to a spatial cluster by a user are logged. Hence for each user, their behavior in visiting a certain cluster is tracked.
- The frequency of visits to each cluster by any User is also taken into account when it comes to defining the cluster profile of a user

Step 4: Community Initialization: using the cluster profile of users.

- We cluster users using a similarity measure in a manner analogous to DBSCAN.
- A random user, u, is picked and the user’s top-k similar users are found. If the kth user has a greater similarity with the user then they are formed as a cluster.
- Top-k users are then pushed into a queue.
- Each user is picked for further expansion of the community.
- Repeated until the queue gets empty.

Step 5: Calculating Community Profiles of various Locations

- Users with similar behavior will tend to visit a similar geo-cluster number of times.
- The number of visits made by a user to geo-cluster calculated.
- A similarity between the two users is obtained.

Step 6: Determining Clusters based on the community profile of Locations.

- User clusters and geo clusters are combined together by calculating community profile which is the number of visits made by user cluster C i to each place p i .

- Depending upon the threshold value pre-decided according to the use case, if a place is grouped under a certain spatial cluster.
- Check to see if it passes the threshold value for being categorized under a particular social cluster then
- That place and be finally clubbed with similar coordinates to form a geo-social cluster.

Data Description

There are two Datasets found ideal for our implementation: Gowalla and Brightkite.

Gowalla contains 196,591 users with 6,442,892 check-ins performed on 1,280,969 places over a period from February 2009 to October 2010.

Brightkite contains 58,228 users with 4,491,143 check-ins from April 2008 to October 2010 across 772,783 places.

The data input to the algorithm is a set of check-ins. A check-in is a tuple (u, x, y, t), where u represents the user ID of the person generating the visit, x, y are the location's spatial coordinates, and t is the timestamp at which the visit occurred.

Sample Gowalla Dataset

[user]	[check-in time]	[latitude]	[longitude]	[location id]
196514	2010-07-24T13:45:06Z	53.3648119	-2.2723465833	145064
196514	2010-07-24T13:44:58Z	53.360511233	-2.276369017	1275991
196514	2010-07-24T13:44:46Z	53.3653895945	-2.2754087046	376497
196514	2010-07-24T13:44:38Z	53.3663709833	-2.2700764333	98503
196514	2010-07-24T13:44:26Z	53.3674087524	-2.2783813477	1043431
196514	2010-07-24T13:44:08Z	53.3675663377	-2.278631763	881734
196514	2010-07-24T13:43:18Z	53.3679640626	-2.2792943689	207763
196514	2010-07-24T13:41:10Z	53.364905	-2.270824	1042822

BrightKite and Gowalla provide an API to access the publicly available data directly. We have the (anonymized) user identifier, the location identifier, the timestamp, and the GPS coordinates where the check-in was made for each check-in. In the datasets, they refer to the GPS coordinates of the venue itself.

	SF_B	NY_B	LA_B	SF_G	NY_G	LA_G
number of users	525	494	371	2,203	1,280	690
number of check-ins	66,593	61,607	63,923	340,366	136,548	79,616
number of locations	12,929	13,592	11,329	15,673	4,074	2,695

Table: Number of users and locations in the cities: San Francisco, New York, and Los Angeles. The subscript denotes the initial of the name of the dataset (Brightkite or Gowalla).

Note that, given the nature of our task, identifying users from check-ins scattered worldwide may be considered an almost trivial task due to the sparsity of the location information and the lack of substantial overlap between different users in their check-ins habits.

We suggest a set of techniques to identify a user given a series of check-ins data. Let C = {c1 . . . cn} denote a set of check-ins. In our dataset, each check-in ci is labeled with a user identifier u_idi, a location identifier l_idi, a timestamp ti, and a GPS point pi indicating where the user performed the check-in.

We propose to solve the task by using location data at different levels of granularity. We use both the trajectory of high-resolution GPS coordinates visited by users and the frequency of visits to the various locations. We conclude the section by introducing a simple yet effective way to measure the identification task's complexity over a given dataset.

Dataset statistics	
Nodes	196591
Edges	950327
Nodes in largest WCC	196591 (1.000)
Edges in largest WCC	950327 (1.000)
Nodes in largest SCC	196591 (1.000)
Edges in largest SCC	950327 (1.000)
Average clustering coefficient	0.2367
Number of triangles	2273138
Fraction of closed triangles	0.007952
Diameter (longest shortest path)	14
90-percentile effective diameter	5.7
Check-ins	6,442,890

Step-by-step implementation with data

Geo-social clustering is straightforward if the set of communities is known. A community is a set of users who have similar interests in visiting places. In other words, if user u visiting a geo-social cluster C increases the chances of user u′ also seeing C, then they are part of the same community. However, we do not have access to the underlying social network and mine organizations. We need to mine communities from just the check-ins. Towards that goal, we base our inference procedure on the following assumptions.

There is an unobserved social network with well-defined communities of users. Connectivity among users in the community is dense and across communities is sparse.

A recent work called C -iC exists in mining communities from user activity data. Each activity is represented by a tuple ⟨u, i, t⟩, where i denotes a particular action such as purchasing a product, sharing a photo, etc. C-IC builds on the assumptions that two users u1,u2 are likely to be part of the same community if u1 is performing an activity i increases the chances of u2 performing i as well. Our problem maps to this same exercise where represents the place ρ∈S visited. We, therefore, adopt the C-IC model for our question of geo-social clustering.

1. **Data Analysis**

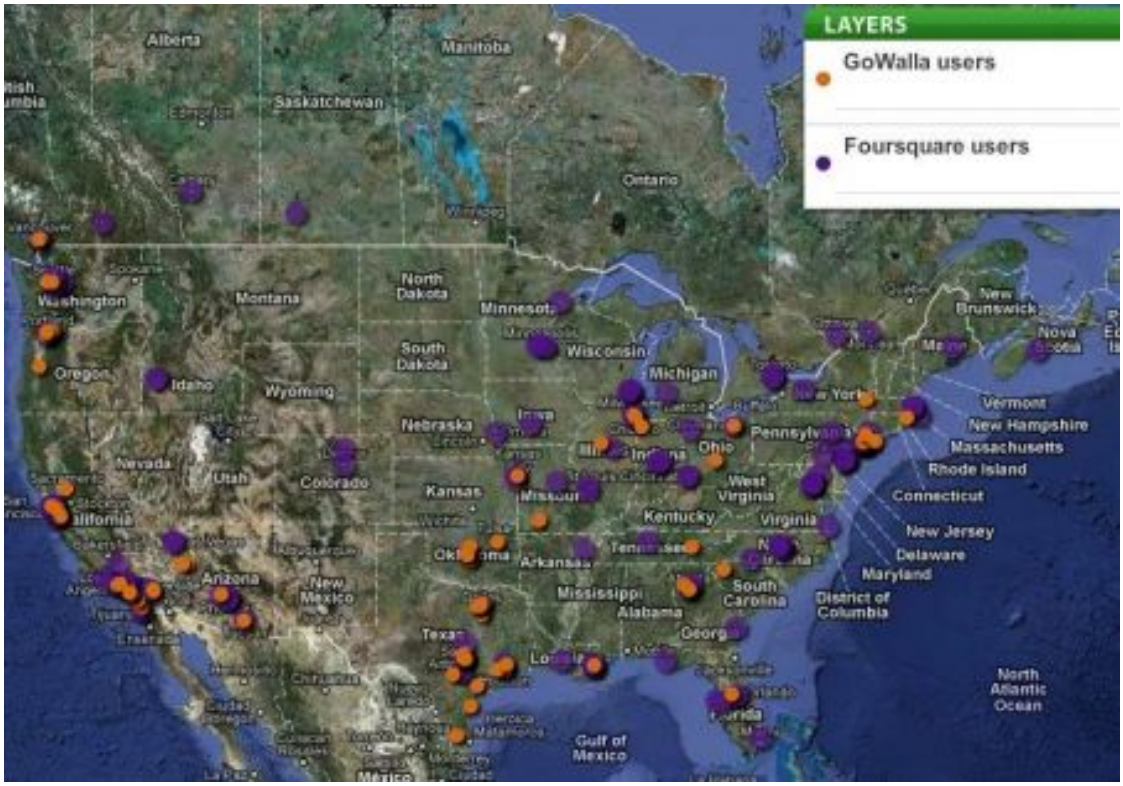
   The exploratory data analysis on the Telecom dataset consisted of:

   - Column description
   - Check for missing data
   - Data graphical analysis

   Column description:

   This dataset contains 6,442,892 rows with 5 columns.

   1. User_ID: User Id
   2. Timestamp: At what time user check in to the place
   3. Latitude: Latitude of the place
   4. Longitude: Longitude of the place
   5. spot_ID: location Id



Data graphical analysis:

Plotted check-ins of the users for a particular place in a map for a Dataset used.

Check for missing data:

Many datasets contain missing values for various reasons. This is also true in our scenario. Different attribute values in the tuple such as x-coordinate or y-coordinate, the timestamp may be missing from several entries. They are often encoded as NaNs, blanks, or any other placeholders. Training a model with a dataset with many missing values can drastically impact the machine learning model's quality.

Some algorithms consider that all values are numerical and have and hold a significant amount. One way to handle this problem is, get rid of the observations that have missing data. However, this results in risking losing data points with valuable details. A better strategy would be to assign the missing values. In other words, we had to infer those missing values from the current part of the data. Mainly there are three main types of missing data:

- Missing completely at random (MCAR)

- Missing at random (MAR)
- Not missing at random (NMAR)

In one of the Dataset, there are some missing values of the user check-ins.

The k nearest neighbor is an algorithm that is used for simple classification. The algorithm uses 'feature similarity' to predict the values of any new data points. This means that the original point is assigned a value based on how closely it resembles the training set's points. This can be very useful in making predictions about the missing values by finding the k's closest neighbors to the observation with missing data and then imputing them based on the neighborhood's non-missing values.

❖ How does it work?

It creates a basic mean impute then uses the resulting complete list to construct a KDTree. Then, it uses the resulting KDTree to compute the nearest neighbors (NN). After it finds the k-NNs, it takes the weighted average of them.

Pros of k means:

It can be much more accurate than the mean, median, or most frequent imputation methods.

On observing the pros of applying k means for filling missing values, the dataset's tuples were refined using this approach wherever there was missing or incorrect data. Hence, preprocessing was done to the dataset. More details are mentioned in the next section.

Data Preprocessing

Columns are unnamed in the Dataset (Index labels are given to the dataset), so column names are assigned to the dataset as userId, timestamp, latitude, longitude, and spotId.

The missing values are filled by taking an average of the nearest place coordinates of the dataset. The column of the spot Id is an unsupported format, so it is converted into numerical data.

Data manipulation using the clustering model

From the formulation of our geosocial clusters, if two points in a geosocial cluster are density connected, they are also associated with the geographical world. This property lies at the core of our initialization procedure.

- After geo-clusters are known labels of clusters are added to the original Dataframe.
- Using pivot function the rows of cluster labels are made as columns and the number of visits of the user to a cluster is obtained.
- The new data frame consists of geo-cluster labels and location coordinates.

Implementation using tools (Screenshots)

1. **Modifying the column names**

```
In[47]:
df1 = pd.read_csv('../input/Gowalla_totalCheckins.txt', sep='\t', header=None,nrows = 35000)
df1[:3]
```

Out[47]:

	0	1	2	3	4
0	0	2010-10-19T23:55:27Z	30.235909	-97.795140	22847
1	0	2010-10-18T22:17:43Z	30.269103	-97.749395	420315
2	0	2010-10-17T23:42:03Z	30.255731	-97.763386	316637

```
In[49]:
df1.columns = ['userid','timestamp','latitude','longitude','spotid']
df1[0:3]
```

Out[49]:

	userid	timestamp	latitude	longitude	spotid
0	0	2010-10-19T23:55:27Z	30.235909	-97.795140	22847
1	0	2010-10-18T22:17:43Z	30.269103	-97.749395	420315
2	0	2010-10-17T23:42:03Z	30.255731	-97.763386	316637

2. Finding Geo-clusters using DBSCAN algorithm

In[60]:

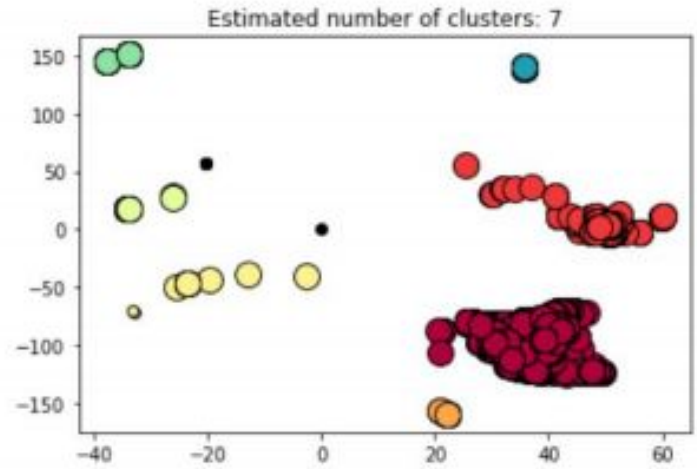
```
## from scipy.spatial.distance import pdist, squareform
from haversine import haversine, Unit
#distance_matrix = squareform(pdist(df, (lambda u,v: haversine(u,v))))
from sklearn.cluster import DBSCAN
import sys
data = df1[["latitude", "longitude"]]
data = data.values.astype("float32", copy = False)

dbsc = DBSCAN(eps = 25, min_samples = 5).fit(data)
core_samples_mask = np.zeros_like(dbsc.labels_, dtype=bool)
core_samples_mask[dbsc.core_sample_indices_] = True
labels = dbsc.labels_

#data['c_labels'] = labels

n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
```

3. Plotting the geo clusters using matplotlib

```
import matplotlib.pyplot as plt

# Black removed and is used for noise instead.
unique_labels = set(labels)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

    xy = data[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)

    xy = data[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)

plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```

4. **A result from the above implementation of clustering places**

```
Estimated number of clusters: 7
Estimated number of noise points: 5
```



+ Code + Markdown

5. **Appending the labels to the existing Dataframe and calculating the similarity between the users to form community clusters**

```
test1['countno'] = 1
columns_to = ['countno']
test2 = test1.groupby(['userid','c'])[columns_to].count().reset_index()
test2
df3 = test2.pivot_table('countno',['userid'],'c')
df3
df4 = df3.fillna(0)
df4
```

Out[83]:

c	-1	0	1	2	3	4	5	6
userid								
0	0.0	225.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	12.0	0.0	0.0	0.0	0.0	0.0
2	0.0	2100.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	211.0	0.0	14.0	0.0	0.0	0.0	0.0
5	0.0	50.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...
138	0.0	960.0	0.0	0.0	0.0	0.0	0.0	0.0
139	0.0	150.0	0.0	0.0	0.0	0.0	0.0	0.0
140	0.0	236.0	0.0	0.0	0.0	0.0	0.0	0.0

```
import numpy
def model(D, eps, MinPts):
    labels = [0]*len(D)
    C = 0
    for P in range(0, len(D)):
        if not (labels[P] == 0):
           continue
        NeighborPts = regionQuery(D, P, eps)
        if len(NeighborPts) < MinPts:
            labels[P] = -1
        else:
           C += 1
           growCluster(D, labels, P, NeighborPts, C, eps, MinPts)
    return labels

def growCluster(D, labels, P, NeighborPts, C, eps, MinPts):
    labels[P] = C
    # dataset.
    i = 0
    while i < len(NeighborPts):
        Pn = NeighborPts[i]
        if labels[Pn] == -1:
           labels[Pn] = C
        elif labels[Pn] == 0:
            labels[Pn] = C
```

```
db = Model(eps=25, min_samples=3)
db.fit(df4)

cluster_labels1 = db.labels_
unique_labels = set(cluster_labels1)

# get the number of clusters
num_clusters = len(set(cluster_labels1))
#x =df4
print('Number of clusters: {}'.format(num_clusters))
```

```
Number of clusters: 5
```

6. **User clusters and geo-clusters are grouped using a similarity metric to form geo-social clusters.**

```
cluster_labels1 = dbscan.labels_
unique_labels = set(cluster_labels1)

# get the number of clusters
num_clusters = len(set(cluster_labels1))
x =df4
print('Number of clusters: {}'.format(num_clusters))
```

```
Number of clusters: 2
```

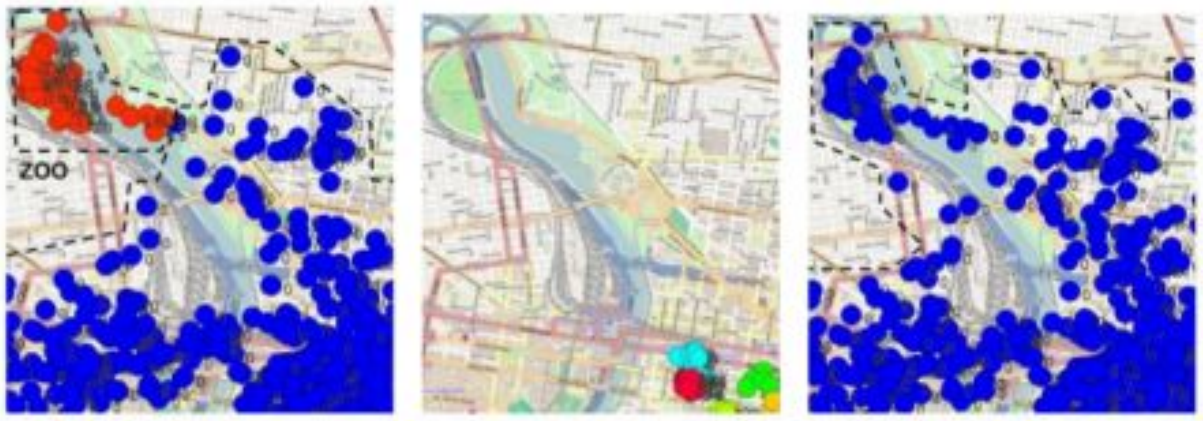
In[91]:

```
df5
```

Out[91]:

c	u_labels	-1	0	1	2	3	4	5	6
0	-1	5.0	19643.0	960.0	0.0	0.0	0.0	29.0	12.0
1	0	0.0	10142.0	83.0	17.0	13.0	13.0	0.0	2.0
2	1	0.0	1042.0	0.0	0.0	0.0	2.0	0.0	0.0
3	2	0.0	44.0	121.0	0.0	0.0	0.0	0.0	0.0
4	3	0.0	2872.0	0.0	0.0	0.0	0.0	0.0	0.0

7. Comparison between the three Algorithms



(a) GeoScop (b) DCPGS (c) DBSCAN

References

1. E. Cho, S. A. Myers, and J. Leskovec, “Friendship and mobility: User movement in location-based social networks,” inSIGKDD, 2011.
2. J. Shi, N. Mamoulis, D. Wu, and D. W. Cheung, “Density-based place clustering in geo-social networks,” inSIGMOD, 2014, pp. 99–110.
3. N. Barbieri, F. Bonchi, and G. Manco, “Influence-based network-oblivious community detection,” inICDM, 2013, pp. 955–960.
4. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise.” in SIGKDD, vol. 96, no. 34, 1996, pp. 226–231.
5. S. Scellato, R. Lambiotte, A. Noulas, and C. Mascolo, “Socio-spatial properties of online location-based social networks,” inICWSM, 2011.
6. L. Backstrom, E. Sun, and C. Marlow, “Find me if you can: Improving geographical prediction with social and spatial proximity,” in 2010, pp. 61–70.
7. V. Kolar, S. Ranu, A. P. Subramanian, Y. Shrinivasan, A. Telang, R. Kokku, and S. Raghavan, “People in motion: Spatio-temporal analytics on call detail records,” in Communication Systems and Networks.
8. D. Pennerstorfer and C. Weiss, “Spatial clustering and market power: Evidence from the retail gasoline market, ”Regional Science and UrbanEconomics, vol. 43, no. 4, pp. 661–675, 2013.
9. A. Diker and E. Nasibov, “Estimation of traffic congestion level via fn-dbscan algorithm by using GPS data,” in Problems of Cybernetics and informatics, 2012, pp. 1–4. 19
10. P. Banerjee, S. Ranu, and S. Raghavan, “Inferring uncertain trajectories from partial observations,” inICDM, 2014, pp. 30–39.
11. I. Ramalho Brilhante, M. Berlingerio, R. Trasarti, C. Renso, J. A. F.de Macedo, and M. A. Casanova, “Come together: Discovering communities of places in mobility data,” in MDM, 2012, pp. 268–273.
12. S. Scellato, A. Noulas, and C. Mascolo, “Exploiting place features in link prediction on location-based social networks,” inSIGKDD, 2011.
13. H. Pham, C. Shahabi, and Y. Liu, “Ebm: An entropy-based model to infer social strength from spatiotemporal data,” in SIGMOD, 2013.