Analysis of social media trends for planning Network resources

Shubham Tiwari ⟨*shubham.tiwari@partner.samsung.com*⟩

ABSTRACT

This project focuses on developing an model using ML that can support prediction of network resources demand based on social media trend analysis in specific location. It aims at overcoming limitations of traditional planning methods and provides approach that can support prediction of network demand based on social media trend analysis in specific location and can thus optimize the cell on/off policy based on real demand. Simulation results are expected as outcome of this execution.

1. Introduction

Often operators are inclined towards reducing their CAPEX (Capital expenditures) and OPEX (operating expenses) by better planning of their network resource usage such as power, radio resources etc. At the same time they must provide high availability and high quality network experience to their user. Often there is a trade off between these two. When demand of network resources increases suddenly in certain area, it often leads to frequent failures of network leading to high OPEX for operators and bad user experience.

Demand of network is high specially in situations such as general election, cricket matches, tournaments, riots, spread of medical emergency, religious procession etc and as this is concentrated demand, it puts sudden high pressure on the otherwise fixed capacity networks.

The proposed idea provided novel solution to intelligently detect the network resource requirements and planning resources when required most.

2. Related Work

Authors in [1] propose a Machine Learning based network planning tool. Data is collected from user-equipment (UE), which may be located at any point in the network, and QoS is estimated based on data through Supervised ML algorithms (SVM, k-NN, NN, DT). The proposed tool is evaluated for two use-cases: (1) small cell deployment in a dense indoor network (2) timely face a detected fault in a macro-cell network. Authors in [2] show that Twitter data can be used to reliably estimate spatial traffic density distribution, and propose a method based on `Voronoi Estimation` [3] to estimate the spatial density, and a queueing theoretic model [4] to estimate the Key Performance Indicators (KPIs). Authors in [5] leverage LDA for topic modelling of Tweets and hierarchical clustering (unsupervised ML), `text2vec` (supervised ML) for predicting the polarity (sentiment) of tweets on the timeline of Mobile Network Operators (MNOs). Article [6] presents a survey of event detection techniques using Twitter data stream. Events are classified into two types: Unspecified events and specified events. In case of detecting unspecified events, no prior information about the event is available. Therefore detection of events in this case requires detecting similarity in trends and grouping them together. In case of specified event detection, information/features about the event are provided by the user. The type of event detection can also be divided into two types: `RED` (Retrospective Event Detection) and `NED` (New Event Detection). `RED` involves detecting previously unidentified events using historical tweet data. `NED` involves detection of new events using Twitter stream in real time. Our use-case can be classified as a `specified NED`. Since we know the bounds of the geographical region that we are interested in, the event is `specified`, and since we are detecting the event in real time using Twitter stream, the type of event detection is `NED`. The survey describes a process of a detecting local geo-social events, by modelling regularities in crowd behavior estimated over a period of time. The geographical area is divided into Regions of Interest (ROIs) using k-means clustering. Irregularities in crowd behavior are detected by comparing the real time Tweet data stream with the estimated behavior.

3. Current Work

Our work focuses on developing an ML based network planning tool taking inspiration from [1], leveraging Twitter data for estimating crowded areas and provisioning cellular resources in those areas. Historical twitter data for the geographical area of

interest is used to estimate the crowd behavior in the region. Using density based clustering, the crowd density can be estimated in a specific region for a regular day. Thresholds for a crowd being an outlier is then calculated. If the number of people in the region is above the threshold value, then the cluster is considered to be an irregular crowd.

4. Streaming Twitter data

`Twitter API` allows streaming of tweets in real time, searching recent tweets, and downloading batch historical tweets. Streaming of tweets in real time can be done with a developer account. However, the API allows allows only a sample of the tweets to be streamed. Actual percentage of the tweets streamed depends on the users request and amount of traffic. Streaming almost 100% of the tweets requires subscribing to `Twitter Firehose`, which is an enterprise feature. The huge quantity of streamed twitter data obtained through `Twitter Firehose` also requires the end user to have sufficient resources to process and store the data. By searching through the `Twitter API`, some of the most recent tweets matching the search criteria can be obtained. However, that may not be sufficient for research purposes. Downloading of batch historical tweets is also an enterprise feature.

Numerous streamed Twitter datasets are available online. For the current work, we use a 12 day stream of geo-located tweets within Tokyo [7].

5. Methodology

5.1. Dataset

The twitter dataset in [7], is supposed to consist of 200,000 geo-tagged tweets for 10 days. However a lot of the tweets lack geographical data. Table 1 shows the number of actual geo-tagged tweets on each day. Days `2016-04-16` and `2016-04-17` clearly have a larger number of tweets as compared to the other days. This indicates that those days were of special significance in Tokyo, and it is expected that there would be outlier crowds on these days. Hence, days `2016-04-16` and `2016-04-17` consist of the test set. Day `2016-04-12` has a very low number of tweets due to very short period of streaming of tweets, and is therefore dropped from the dataset. Rest of the days consist of the training set.

Day	# Tweets
2016-04-21	252
2016-04-20	299
2016-04-19	353
2016-04-18	346
2016-04-17	456
2016-04-16	513
2016-04-15	293
2016-04-14	370
2016-04-13	359
2016-04-12	26

Table 1. # Tweets on each day

5.2. Method I: Density based clustering

Density based clustering is described in [8]. A circular area with `Tokyo Station` ($x_0 = (35.681308, 139.767127)$) as the center and radius $d = 69752.35$ meters is chosen as the area under consideration for detecting crowds. For each hour, ϵ is estimated by Eq. 1:

$$\epsilon = \frac{1}{n}\sum_{i=1}^{n} dist_i \tag{1}$$

where n is the number of geo-tagged tweets in the hour and $dist_i$ is the minimum distance of point x_i from all other points in that hour. Using ϵ estimated from Eq. 1, $MinPts$ is estimated is estimated by Eq. 2:

$$MinPts = \frac{1}{n}\sum_{i=1}^{n} NumPts_i \tag{2}$$

where $NumPts_i$ is the number of points inside a radius of ϵ from each point x_i. The estimated ϵ and $MinPts$ are then chosen as arguments to the `DBSCAN` clustering algorithm.

5.3. Method II: Estimate BS resource utilization using queueing theory

Authors in [4], model the cellular wireless network traffic flow using an M/M/1/K-PS (processor sharing) queueing model. The model estimates Key-Performance Indicators (KPIs) such as: User specific KPIs - (1) expected flow throughput, (2) expected flow sojourn time and Network specific KPIs - (1) BS resource utilization (2) expected probability of blocking a data flow request (3) expected mean number of concurrently active data flows. For the current use-case, given the user distribution in the region considered, our objective is to predict an optimal ON/OFF configuration of eNBs. This can be achieved by the estimating the probability of each cell serving a flow given the user distribution in the region. The queueing model described in [4] can be used estimate the BS resource utilization, which is equal to the probability of each cell serving a flow.

Rest of the Section is organised as follows: Section 5.3.1 discusses the methods of estimating user density distribution from the streamed twitter data, Section 5.3.2 discusses the system model and Section 5.3.3 illustrates the algorithm used to estimate the flow probabilities.

5.3.1. User density estimation

Since we aim to model the cellular network with a M/M/1/K-PS queue, we must estimate the intensity of the Poisson arrivals. Authors in [3] propose a method to estimate the intensity of a planar Poisson process by constructing a Voronoi Tessellation, called Voronoi estimation. Let $\mathcal{L}$ be a compact plane and let $P = \{p_i\}_{i=1}^{n}$ be a realization of a planar Poisson process. Then the Voronoi Tessellation $T = \{C_1, C_2, ..., C_n\}$ is a partition of the plane $\mathcal{L}$ such that each point C_i is closer to the point p_i than any other point in the plane. For any location $y_i \in T$, the Voronoi estimate $\hat{\lambda}_i$ is defined as $\hat{\lambda}_i = 1/\mu(C_i)$, where $\mu(.)$ is the appropriate Lebesgue measure. Each Voronoi region y_i is obtained by the intersection of half-planes, and is therefore a convex polygon. $\mu(C_i)$ in this case, is simply the area of the polygon representing the region. Authors in [2] show that there is a high correlation between the traffic intensities measured by User Equipments (UE) and scaled user densities estimated by the Voronoi estimator using the Twitter stream.

For the current work, we use the Voronoi estimator to estimate the Poisson arrival intensities. The tweet geo-tags represent a set of points from the tweets were posted. Voronoi estimation over these set of points is an estimate of the user densities. Given average `Think Time` (t_u) for each user, and the user density estimate at the point u as $d(u)$ the arrival rate at the point u is $d(u)/t_u$. The total arrival rate for a region $\mathcal{L}$ can be obtained as follows:

$$\lambda_{\mathcal{L}} = \int_{\mathcal{L}} \frac{d(u)}{t_u} du \tag{3}$$

From the implementation perspective, the region $\mathcal{L}$ can be divided into m equal part,s each of area δu, and the density at the center of each part i can be considered to be the density of the entire part. The integral can be approximated by the summation over all the parts of the region as follows:

$$\lambda_{\mathcal{L},m} = \sum_{u_i \in \mathcal{L}} \frac{d(u_i)}{t_u} \delta u, \;\; i \in \{1, .., m\} \tag{4}$$

As the value of m is made larger, the area of the region δu becomes smaller and $\lambda_{\mathcal{L},m}$ approaches $\lambda_{\mathcal{L}}$: $\lim_{m \to \infty} \lambda_{\mathcal{L},m} = \lambda_{\mathcal{L}}$

5.3.2. System Model

We consider a region $\mathcal{L}$ composed of N cells, where region covered by cell i is denoted by $\mathcal{L}_i$. Each UE assumed to be connected to a unique BS, such that $\mathcal{L}_i \cap \mathcal{L}_j = \phi \forall i, j \in 1, .., N$ and $\bigcup_{i=1}^{N} \mathcal{L}_i = \mathcal{L}$.

Let the configuration of each BS (ON/OFF) be represented by an N bit string $\xi = \{0, 1\}^N$ and define $I_\xi : \mathbb{N} \to \mathbb{N}$ such that $I_\xi(i) = 1$ if i^{th} bit of ξ is 1 else $I_\xi(i) = 0$. Also, let $\chi_i \subset \{1, .., N\}$ be the set of BS interfering with the frequency of BS serving $\mathcal{L}\rangle_i$ The SINR at each point u in $\mathcal{L}_i$ is given by:

$$\gamma_i(u, \xi) = \frac{p_i(u)}{\sum_{j \in \chi_i} I_\xi(j).p_j(u) + N_o} \tag{5}$$

where $p_k(u)$ is the power received at point u from the BS serving the region $\mathcal{L}_i$ and N_o is the internal noise temperature. The maximum achievable rate $c_{i,\xi}(u)$ at point $u \in \mathcal{L}_\rangle$ is given by:

$$c_i(u, \xi) = aBmin\{log_2(1 + b\gamma_i(u, \xi)), c_{max}\} \tag{6}$$

where B is the bandwidth, a is the bandwidth efficiency, b is the SINR efficiency and c_{max} is the maximum data rate achievable, and depends on the wireless hardware at hand. Using the streamed Twitter data, and method discussed in 5.3.1, user density $d(u)$ at each point $u \in \mathcal{L}$ can be estimated. Normalized user density $d_i(u)$ for region $\mathcal{L}_i$ is given by:

$$d_i(u) = \frac{d(u)}{\int_{\mathcal{L}_i} d(u)du} \tag{7}$$

For implementation purposes, the integration can be discretized, similar to the procedure followed in Section 5.3.1. The average rate $C_{i,\xi}$ in the region $\mathcal{L}_i$ is given by:

$$C_i(\xi) = \left[\int_{\mathcal{L}_i} \frac{d_i(u)}{c_i(u,\xi)}\right]^{-1} \tag{8}$$

Assuming that each flow is of size Ω Mbps on an average, the mean service rate μ_i offered by the BS serving the region $\mathcal{L}_i$ is given by:

$$\mu_i(\xi) = \frac{C_i(\xi)}{\Omega} \tag{9}$$

Therefore, the load of BS i, ρ_i, serving the region $\mathcal{L}_i$ is given by:

$$\rho_i(\xi) = \frac{\hat{\lambda}_{\mathcal{L}_i}}{\mu_i(\xi)} \tag{10}$$

For systems with admission control, the BS resource utilization η_i for the BS serving the region $\mathcal{L}_i$ is given by:

$$U_i(\xi) = \rho_i(\xi)\frac{1-\rho_i^K(\xi)}{1-\rho_i^{K+1}(\xi)} \tag{11}$$

5.3.3. Algorithm

Let η_i be the resource utilization for BS serving the area $\mathcal{L}_i$. Let $\eta = (\eta_1, ..., \eta_N)$. Then the SINR in Eq. 5 can be written as:

$$\gamma_i(u,\eta) = \frac{p_i(u)}{\sum_{j\in\chi_i} \eta_i.p_j(u) + N_o} \tag{12}$$

$$c_i(u,\eta) = aBmin\{log_2(1+b\gamma_i(u,\eta)), c_{max}\} \tag{13}$$

$$d_i(u) = \frac{d(u)}{\int_{\mathcal{L}_i} d(u)du} \tag{14}$$

$$C_i(\eta) = \left[\int_{\mathcal{L}_i} \frac{d_i(u)}{c_i(u,\eta)}\right]^{-1} \tag{15}$$

$$\mu_i(\eta) = \frac{C_i(\eta)}{\Omega} \tag{16}$$

$$\rho_i(\eta) = \frac{\hat{\lambda}_{\mathcal{L}_i}}{\mu_i(\eta)} \tag{17}$$

$$U_i(\eta) = \rho_i(\eta)\frac{1-\rho_i^K(\eta)}{1-\rho_i^{K+1}(\eta)} \tag{18}$$

Given resource utilization vector η_{k-1}, Eq. 17 can be used to obtain a resource utilization vector η_k in the following way:

$$\eta_k = (U_1(\eta_{k-1}), .., U_N(\eta_{k-1})) \tag{19}$$

Given a initial resource utilization vector η_o, a sequence of resource utilization vectors $\{\eta_k\}_{k=0}^{\infty}$ can be constructed, which converges to a unqiue fixed point [4]. By using fixed-point iteration, we can obtain a sufficiently accurate solution $\hat{\eta}$ to Eq. 20:

$$\eta = (U_1(\eta), .., U_N(\eta)) \tag{20}$$

To illustrate the fixed-point iteration method, we define $\|.\|_{max}$ for a vector x as follows:

$$\|x\|_{max} = \max_i x_i \tag{21}$$

where x_i is the value of the vector in the i^{th} dimension. Algorithm `compute_resource_util_iter` illustrates the function `solve()` which computes the solution to the Eq. 20.

Algorithm 1: compute_resource_util_iter

function compute_res_util (η);
Input : resource utilization vector η
Output : updated resource utilization vector η_u
    $\eta_u = (U_1(\eta), .., U_N(\eta))$
    return η_u
function solve (η_o, tol);
Input : initial resource utilization vector η_o and tolereance tol
Output : final vector η_f after performing fixed-point iteration
    η_{new} = η_o
    do
        η_{prev} = η_{new}
        η_{new} = compute_res_util(η_{prev})
     while ($\|\eta_{new} - \eta_{prev}\|_{max} > tol$)
    return η_{new}

6. Evaluation

The deployment scenario considered for evaluation is the urban macro-cell scenario as specified in [9]. Network topology considered is a hexagonal layout with 19 cells and 3 sites per cell. The `SINR` distribution for the network layout is illustrated in Fig. 1. Detailed simulation parameters are listed in 2. The user distribution considered are geo-tagged tweets for an hour `22:00` (Fig. 4w). Since the users are scattered in large distances, and the network topology accounts for only a small portion of the region centered around the `Tokyo station` (35.681308, 139.767127), the distances were scaled down by a factor of 20. Also, the twitter densities obtained were scaled up by a factor of `1000000`. For practical deployment purposes, the scaling factor can be estimated by collecting data from UEs and comparing them with the Twitter densities.

Table 2. Simulation parameters

Parameter	**Value**
Number of concurrent flows (K)	2
Think Time	2 secs
Flow size (Ω)	2 Mbps
Fast fading margin	-2 dBm
Antenna diversity gain	3 dBm
BS noise figure	-5 dBm
UE noise figure	-9 dBm
Internal noise temperature	-100.81 dBm
Maximum TX power	49 dBm
Bandwidth efficiency (a)	0.63
SINR efficiency (b)	0.4
Bandwidth (B)	20 Mhz
Maximum achievable rate (c_{max})	100.8 Mbps
BS height	32 meters
UE height	1.5 meters
Antenna gain model	$G(\theta,\phi) = -\min[-(A(\theta)+A_e(\phi)), A_m]$
	$A(\theta) = -\min[12(\frac{\theta}{\theta-3dB})^2, A_m]$
	$A_e(\phi) = -\min[12(\frac{\phi-\phi_{tilt}}{\phi_{3dB}})^2, A_m]$
	$-180° \leq \theta \leq 180°, -90° \leq \phi \leq 90°$
Maximum antenna attenuation (A_m)	20 dB
θ_{3dB}	$70°$
ϕ_{3dB}	$15°$
Antenna tilt (ϕ_{tilt})	$0°$
Antenna orientation	$120° sector$
UT Antenna	omni-directional
Path-loss model	$-128.1 + 37.6\log_{10}(d)$, d is in kms
Inter-site distance (ISD)	500 meters
Frequency re-use factor	3

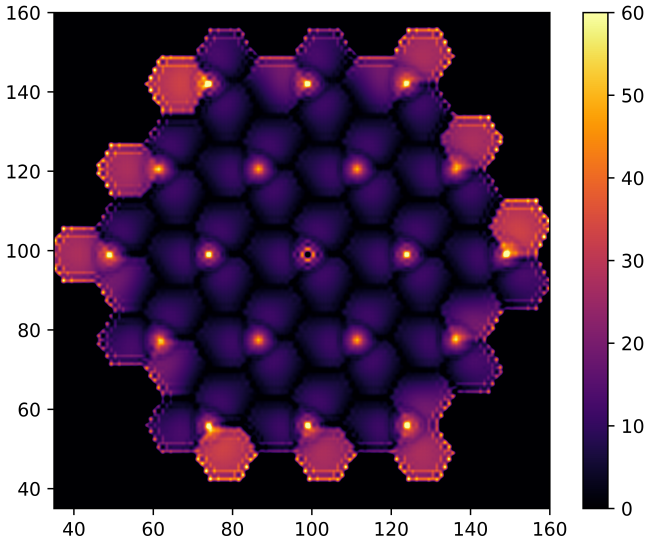
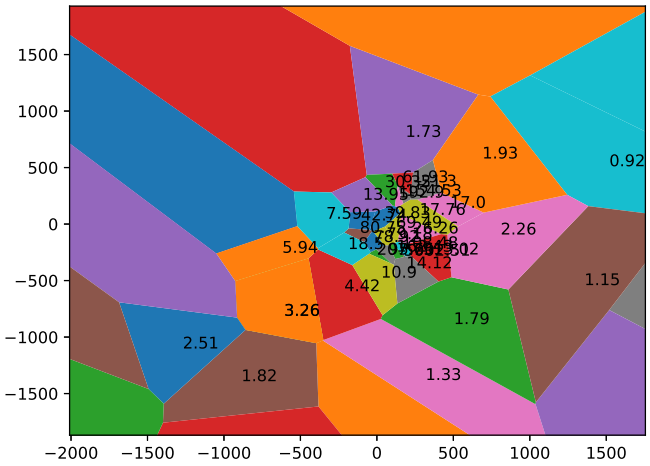
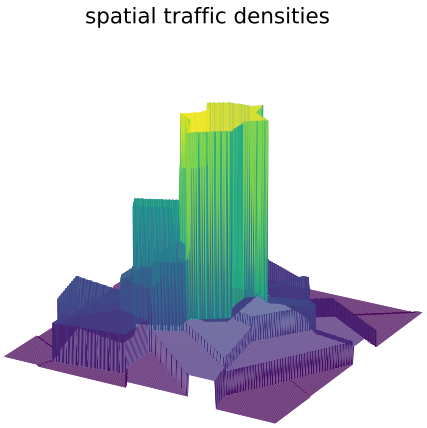


Figure 1. SINR distribution for the urban macro cell setup



(a) Spatial user density obtained through voronoi tessellation



(b) User density distribution in the region

7. Results

7.1. Method I: Density based clustering

The results of `DBSCAN` clustering are shown in Figure 4. (a) - (x) and Figure 5. The scripts used to produce the results can be found in Section 9.

7.2. Method II: Estimate BS resource utilization using queueing theory

The Voronoi estimated user density distribution is illustrated by the Voronoi tessellation in Fig. 2a. The 3D representation of the user densities is shown in Fig. 2b. The `solve()` function defined above, was executed on the user density distribution from the Tokyo twitter stream for the hour `22:00` with the simulation parameters discussed in section 6. The initial resource utilization vector $\eta_o = \{0.5\}^{57}$ and $tol = 0.0001$ was passed as arguments to the function `solve()`. It took 5 iterations of the fixed-point method to obtain the final resource utilization vector η_f representing the resource utilization for each cell. The graph illustrating the convergence of η with the number of iterations is shown in Fig. 3.

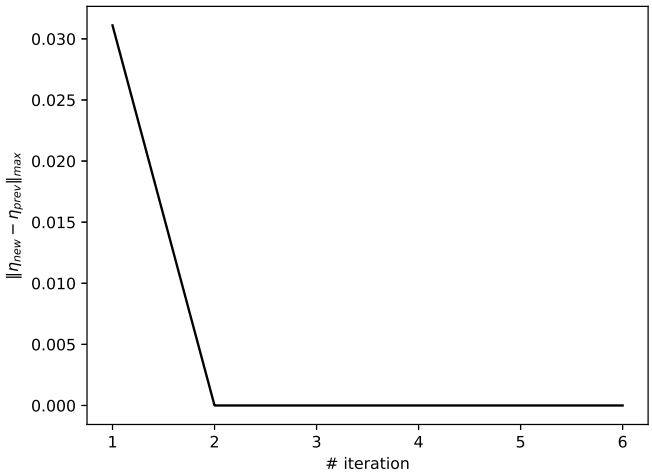


Figure 3. Convergence of resource utilization vector η

The final vector $\hat{\eta}$ obtained is:

[0.2679068723491424, 0.14738280156151176, 0.4283858654943523, 0.02789698995889222, 0.20013948604685552, 0.0411721198177242 15, 0.019325143066753685, 0.14630609925273416, 0.1742814591039783, 0.015863969818302223, 0.001418874936263261, 0.02465615527484906, 0.0018532885554886612, 0.0013126598075483052, 0.01152976340222729, 0.10368015820900348, 0.01276132944279308, 0.04044471049680042, 0.16078839117908209, 0.0688556431687809, 0.05472478687284563, 0.013190305409037538, 0.020617366974863223, 0.012135633177152313, 0.012305321896047026, 0.017619054803135564, 0.016926286827933092, 0.014457557501560993, 0.0018189191810612441, 0.006299476771428023, 0.001052818154215835, 0.0012014739755797543, 0.0011742743033864 49, 0.038389633913922756, 0.0005742605823394885, 0.0012592786374855235, 0.014956801200792878, 0.019360255260458346, 0.009988615347321298, 0.021007312064428223, 0.022612908709256384, 0.01978081356932352, 0.016414056902590186, 0.016446554432346442, 0.020721693604366157, 0.0011034452365043823, 0.0010928887205961003, 0.001099985515518407, 0.0011921018029946112, 0.0012238591657376997, 0.0022347974481551213, 0.04343933367855403, 0.03049534943959724, 0.030576754781262174, 0.017813278346126737, 0.02281481716965216, 0.015654979918181756]

8. Future Work

Future work will involve using the test dataset to evaluate the the density based crowd detection using `DBSCAN`. Allocation of cellular network resources in crowded area will be evaluated through simulations.

References

[1] J. Moysen, L. Giupponi, and J. Mangues-Bafalluy, "A mobile network planning tool based on data analytics," *Mobile Information Systems*, vol. 2017, 2017.

[2] H. Klessig, H. Kuntzschmann, L. Scheuvens, B. Almeroth, P. Schulz, and G. Fettweis, "Twitter as a source for spatial traffic information in big data-enabled self-organizing networks," in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, 2017, pp. 1–5.

[3] C. D. Barr and F. P. Schoenberg, "On the voronoi estimator for the intensity of an inhomogeneous planar poisson process," *Biometrika*, vol. 97, no. 4, pp. 977–984, 2010.

[4] H. Klessig, D. Öhmann, A. J. Fehske, and G. P. Fettweis, "A performance evaluation framework for interference-coupled cellular data networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 2, pp. 938–950, 2016.

[5] K. A. Ogudo and D. M. J. Nestor, "Sentiment analysis application and natural language processing for mobile network operators' support on social media," in *2019 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD)*, 2019, pp. 1–10.

[6] F. Atefeh and W. Khreich, "A survey of techniques for event detection in twitter," *Comput. Intell.*, vol. 31, no. 1, p. 132–164, Feb. 2015. [Online]. Available: https://doi.org/10.1111/coin.12017

[7] "200,000 tokyo geolocated tweets. free twitter dataset," http://www.followthehashtag.com/datasets/200000-tokyo-geolocated-tweets-free-twitter-dataset/, accessed: 2020-06-12.

[8] M. B. Khalifa, R. P. Díaz Redondo, A. F. Vilas, and S. S. Rodríguez, "Identifying urban crowds using geo-located social media data: A twitter experiment in new york city," *J. Intell. Inf. Syst.*, vol. 48, no. 2, p. 287–308, Apr. 2017. [Online]. Available: https://doi.org/10.1007/s10844-016-0411-x

[9] M. Series, "Guidelines for evaluation of radio interface technologies for imt-advanced," *Report ITU*, vol. 638, pp. 1–72, 2009.

9. Scripts

9.1. Twitter Stream

```
import tweepy
import pprint

pp = pprint.PrettyPrinter(indent=4)

consumer_key = "###"
consumer_secret = "###"

access_token = "###"
access_token_secret = "###"

#override tweepy.StreamListener to add logic to on_status
class MyStreamListener(tweepy.StreamListener):
```

```
        def on_status(self, tweet):
                if not ((tweet._json['place'] is None) and (tweet._json['geo'] is None) and (
                        pp.pprint(tweet._json)
                        print(f"{tweet.user.name} -> {tweet.text}")


#auth = tweepy.AppAuthHandler(consumer_key, consumer_secret)
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

api = tweepy.API(auth, wait_on_rate_limit=True,
    wait_on_rate_limit_notify=True)

myStreamListener = MyStreamListener()
myStream = tweepy.Stream(auth = api.auth, listener=myStreamListener)

myStream.filter(track=['COVID19'])
```

9.2. Clustering using DBSCAN

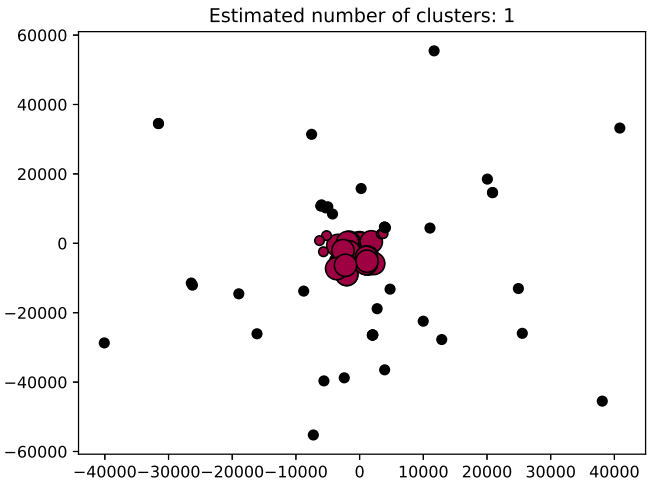
```
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import pairwise_distances

import pandas as pd
import numpy as np
import sys
import os
from datetime import datetime, timedelta
import geopy.distance
import matplotlib.pyplot as plt

from joblib import dump, load
import pickle

data = pd.read_csv('tokyo.csv', sep=',',encoding = 'utf8',lineterminator='\n',low_memory=Fals
data = data.dropna()

# Maximum number of tweets are on 2016-04-16 and 2016-04-17, so drop all rows related to that
# They will act as the test data
data = data.drop(data[data['Date']=='2016-04-16'].index)
data = data.drop(data[data['Date']=='2016-04-17'].index)
# Very low number of tweets on 2016-04-12, so drop it
data = data.drop(data[data['Date']=='2016-04-12'].index)
#-------------------------------------------------------------------------------#

# get date time object
#data['time'] = datetime.strptime(data['Date']+' '+data['Hour'], '%y-%m-%d %H:%M')
data['time'] =  (data['Hour']).apply(lambda x: datetime.strptime(x, '%H:%M'))
data = data.set_index('time')
print('Columns in dataset:')
print(list(data.columns.values))

# printing dataframe rows where date is equal to date we want
max = 0
```

```
date_m = None
print('Numbers of tweets in each day:')
for date in data['Date'].unique():
        n = len(data[data['Date']==date])
        print('{} on {}'.format(n, date))
        if n > max:
                max = n
                date_m = date

print('Max number of tweets in a day: {} on {}'.format(max, date_m))
#print(data['Date'][data['Date']=='2016-04-12'])

#print('Times in a day: ')
#print(data['Hour'][data['Date']=='2016-04-16'].unique())

# Tokyo station: 35.681308, 139.767127
x = 35.681308
y = 139.767127
origin = (x,y)
print('Tokyo station: x={}, y={}'.format(x,y))

#coords_1 = (52.2296756, 21.0122287)
#coords_2 = (52.406374, 16.9251681)

#print(geopy.distance.vincenty(coords_1, coords_2).m)

# find the maximum distance d in the dataset
d = 0
for index, row in data.iterrows():
        coord = (row['Latitude'], row['Longitude'])
        curr = geopy.distance.vincenty(origin, coord).m
        if curr > d:
                d = curr

print('Maximum distance of a geo-tagged tweet from the origin is: {} meters'.format(d))

#quit()

def calc_min(origin, hdf):
        min = float('inf')
        for index, row in hdf.iterrows():
                if origin[0]==row['Latitude'] and origin[1]==row['Longitude']:
                        continue
                coord = (row['Latitude'], row['Longitude'])
                dist = geopy.distance.vincenty(origin, coord).m
                if dist < min:
                        min = dist
        return min

def calc_min_pts(origin, hdf, eps):
        sum = 0
        for index, row in hdf.iterrows():
                if origin[0]==row['Latitude'] and origin[1]==row['Longitude']:
                        continue
                coord = (row['Latitude'], row['Longitude'])
```

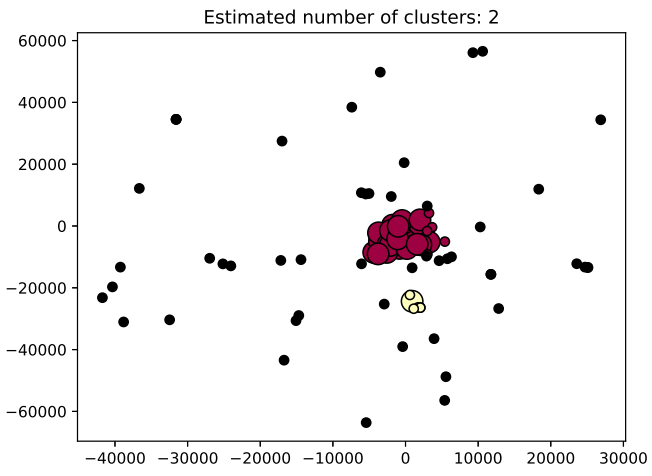
```
                dist = geopy.distance.vincenty(origin, coord).m
                if dist < eps:
                        sum = sum + 1
        return sum

def distance_in_meters(x, y):
    return geopy.distance.vincenty((x[0], x[1]), (y[0], y[1])).m

df = {}
eps_dict = {}
min_pts_dict = {}
cluster = {}
n_clusters = pd.DataFrame(columns=['time', 'n_clusters'])
# sort by dates
#for date in data['Date'].unique():
#       df[date] = data[data['Date']==date]

data = data.groupby(pd.Grouper(freq='60Min'))
for key, item in data:
    #print(key, "\n\n") # data.get_group(key) to get the group values
    hdf = data.get_group(key)
    sum = 0
    for index, r in hdf.iterrows():
        sum = sum + calc_min((r['Latitude'], r['Longitude']), hdf)
    eps = sum/len(hdf.index)
    #print('key: {}, eps={}'.format(key, eps))

    pts = 0
    for index, r in hdf.iterrows():
        pts = pts + calc_min_pts((r['Latitude'], r['Longitude']), hdf, eps)
    min_pts = pts/len(hdf.index)
    print('key: {}, eps={} min_pts={}'.format(key, eps, min_pts))
    eps_dict[key] = eps
    min_pts_dict[key] = min_pts

    # Apply fit transform for the current hour
    X = hdf[['Latitude','Longitude']]
    distance_matrix = pairwise_distances(X, metric=distance_in_meters)
    #print(distance_matrix)
    db = DBSCAN(eps=eps, min_samples=int(min_pts), metric='precomputed')
    db.fit(distance_matrix)
    cluster[key] = db
    # save the trained model on the disk for future use
    dump(db, 'clusters/trains/train' + key.strftime("%H") + '.joblib')
    X.to_pickle('clusters/data/data' + key.strftime("%H") + '.pkl')
    ### Output ###
    core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
    core_samples_mask[db.core_sample_indices_] = True
    labels = db.labels_

    # Number of clusters in labels, ignoring noise if present.
    n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
    n_noise_ = list(labels).count(-1)

    n_clusters = n_clusters.append(pd.DataFrame([[key.strftime("%H:%M"), n_clusters_]], colum
```

```
    print('Estimated number of clusters: %d' % n_clusters_)
    print('Estimated number of noise points: %d' % n_noise_)
#     print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels))
#     print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels))
#     print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels))
#     print("Adjusted Rand Index: %0.3f"
#           % metrics.adjusted_rand_score(labels_true, labels))
#     print("Adjusted Mutual Information: %0.3f"
#           % metrics.adjusted_mutual_info_score(labels_true, labels))
    print("Silhouette Coefficient: %0.3f"
          % metrics.silhouette_score(X, labels))

    # #############################################################################
    # Plot result
     #converting points with reference to x and y
    X = X.astype(float)
    #print('-- after conversion --')
    #print(X['Latitude'])
    X['Latitude'] = X['Latitude'].apply(lambda t: np.sign(t - float(x))*(geopy.distance.vince
    X['Longitude'] = X['Longitude'].apply(lambda t: np.sign(t - float(y))*(geopy.distance.vin
    #X['Latitude'] = X['Latitude'] - x
    #X['Longitude'] = X['Longitude'] - y

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

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

    plt.title('Estimated number of clusters: %d' % n_clusters_)
#     plt.savefig('clusters/plot_'+ key.strftime("%H-%M") +'.png', bbox_inches='tight')
    loc = 'clusters/png/plot_'+ key.strftime("%H-%M") +'.png'
    plt.savefig(loc, dpi=1000, bbox_inches='tight')

    loc = 'clusters/eps/plot_'+ key.strftime("%H-%M") +'.eps'
    plt.savefig(loc, format='eps', dpi=1000, bbox_inches='tight')

    # Clear the plot
    plt.clf()
    plt.cla()
    plt.close()
```

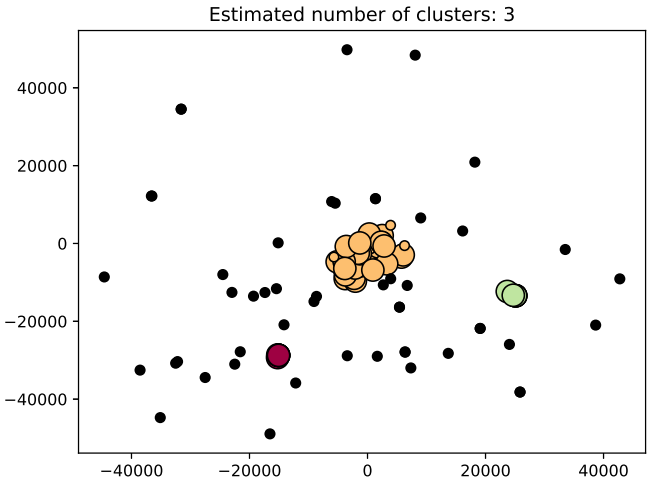
```
# save the cluster data
n_clusters.to_pickle('clusters/nclusterdata.pkl')
# Plot number of clusters as a function of the hour of the day
plt.plot(n_clusters['time'], n_clusters['n_clusters'])
plt.title('Number of clusters in each time of day')
plt.xlabel('hour of the day')
plt.ylabel('number of clusters')
plt.xticks(rotation=45)

#pickle.dump(ax, open("plot.pickle", "wb"))

loc = 'clusters/nclusters.png'
plt.savefig(loc, dpi=1000, bbox_inches='tight')

loc = 'clusters/nclusters.eps'
plt.savefig(loc, format='eps', dpi=1000, bbox_inches='tight')

# Clear the plot
plt.clf()
plt.cla()
plt.close()
```



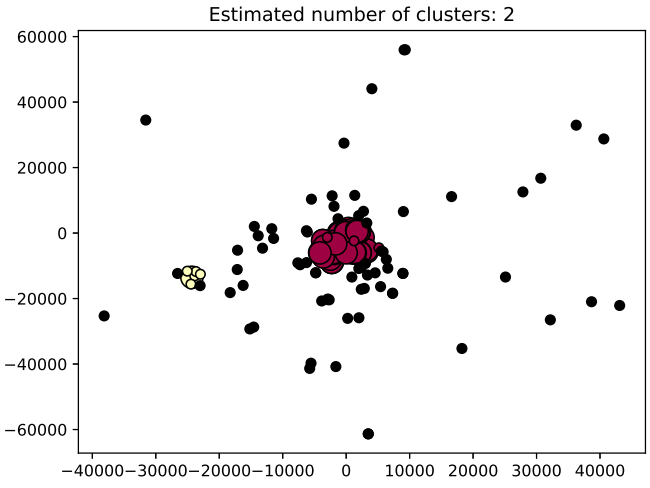
(a) Hour 00:00



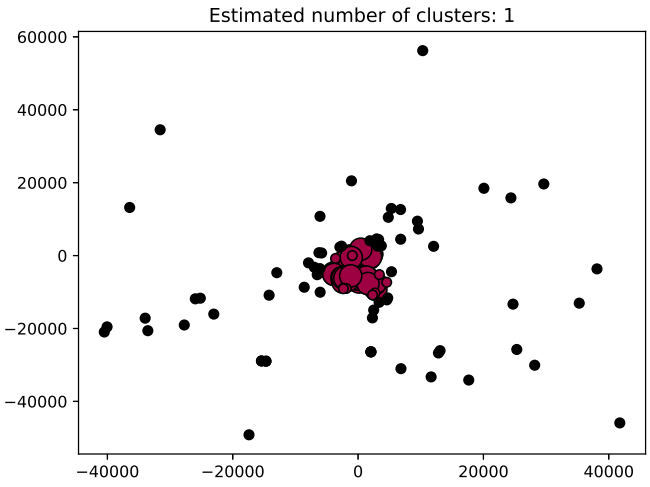
(b) Hour 01:00



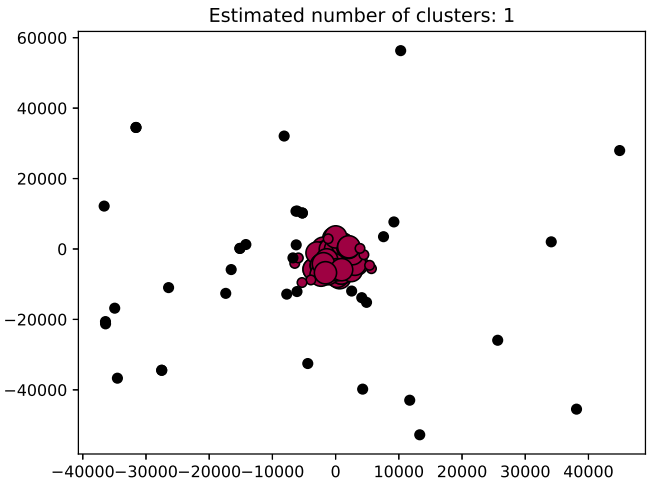
(c) Hour 02:00



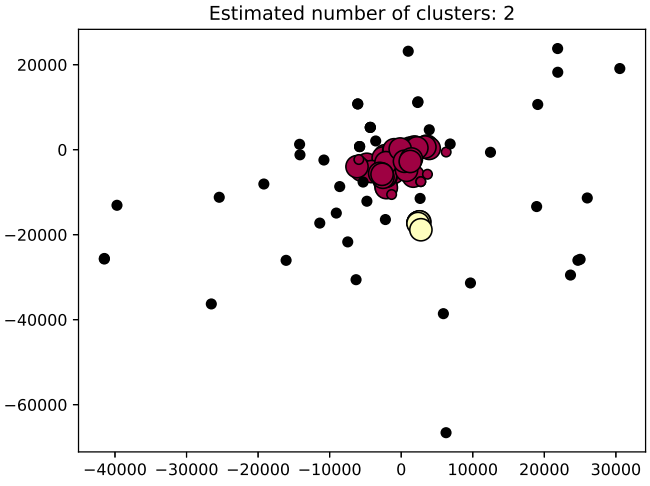
(d) Hour 03:00



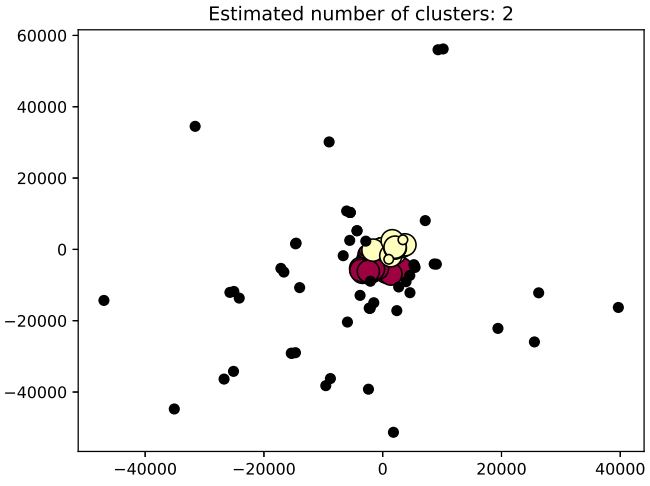
(e) Hour 04:00



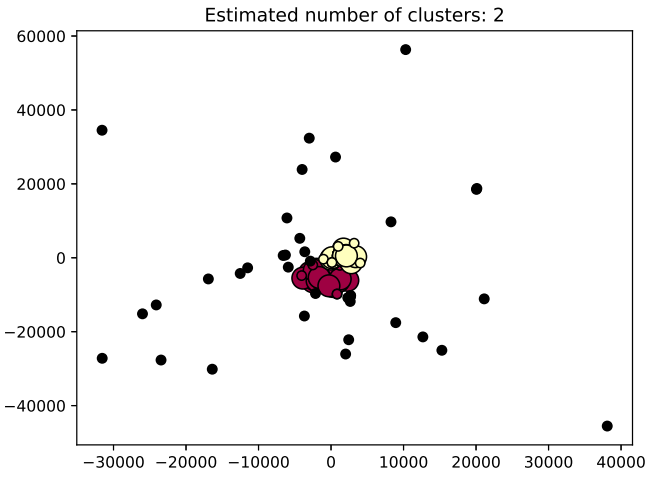
(f) Hour 05:00



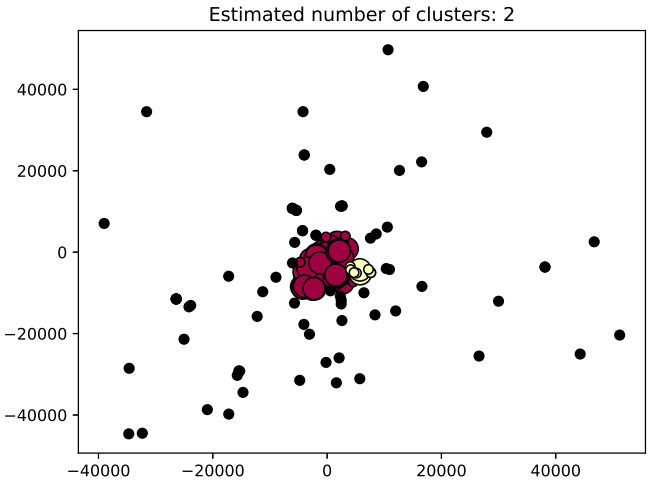
(g) Hour 06:00



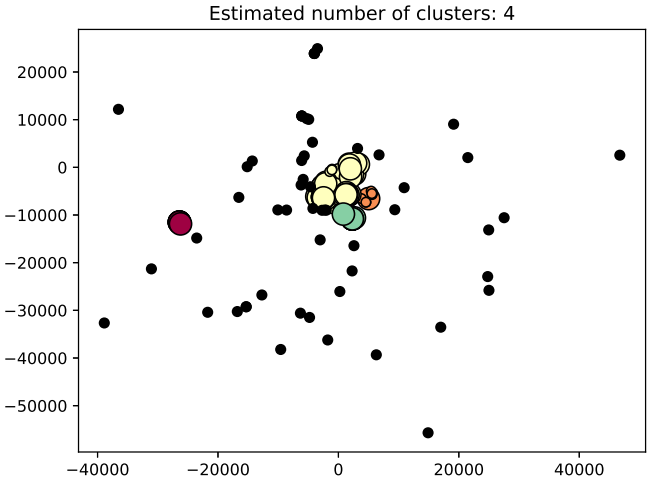
(h) Hour 07:00



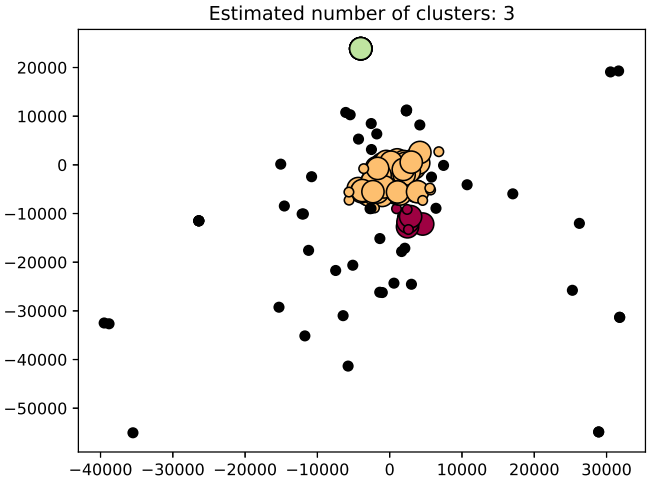
(i) Hour 08:00



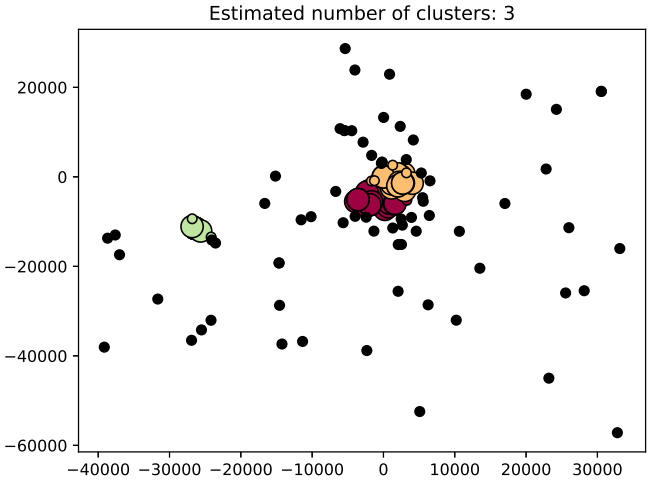
(j) Hour 09:00



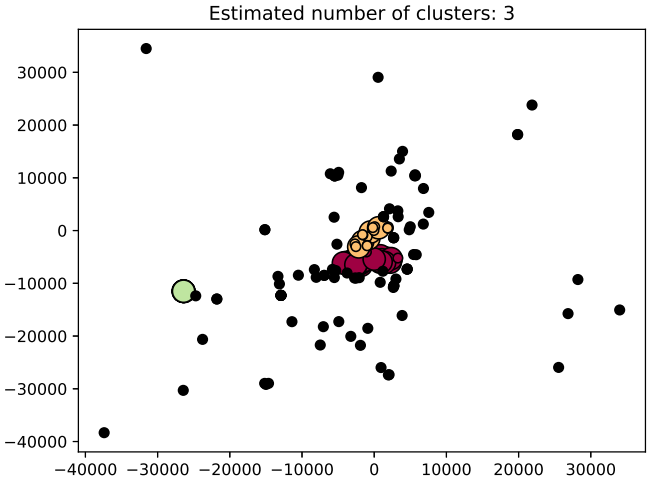
(k) Hour 10:00



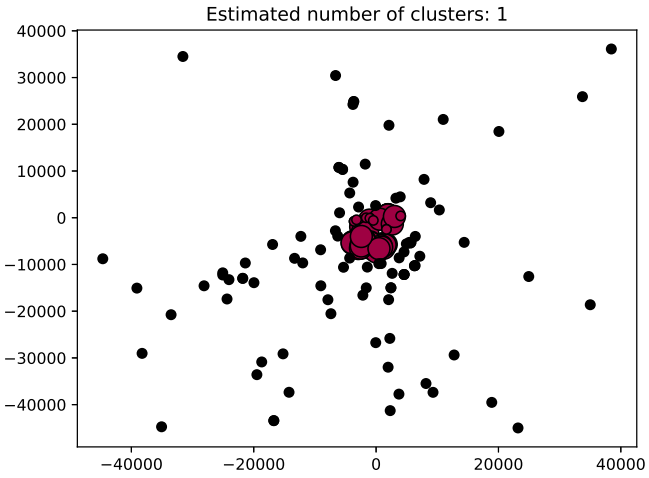
(l) Hour 11:00



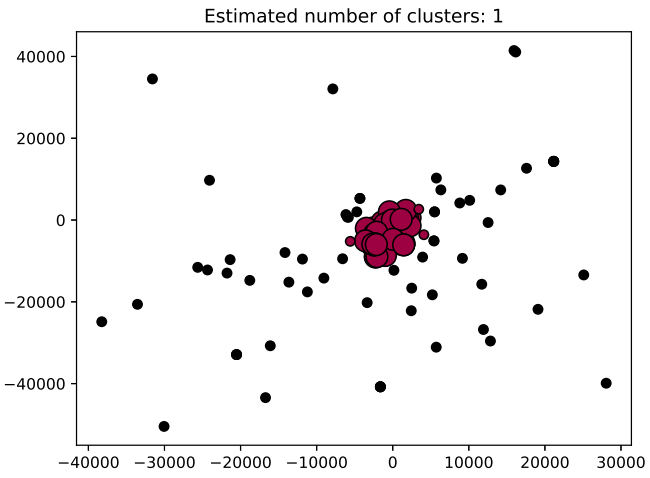
(m) Hour 12:00



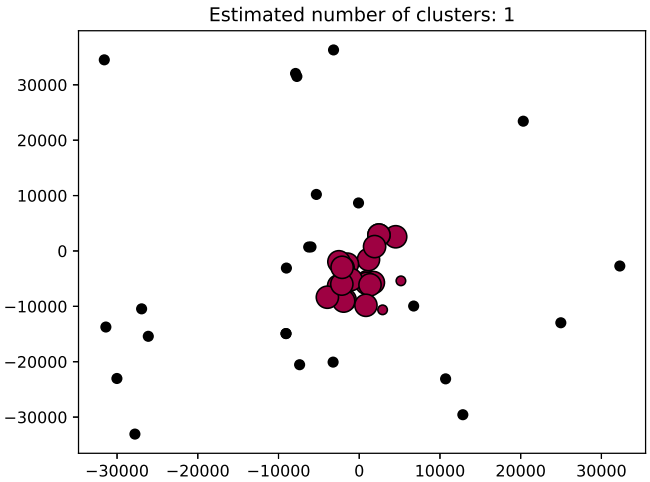
(n) Hour 13:00



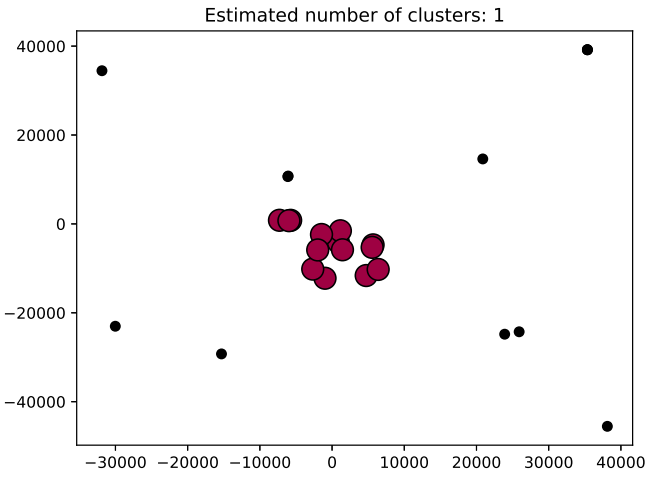
(o) Hour 14:00



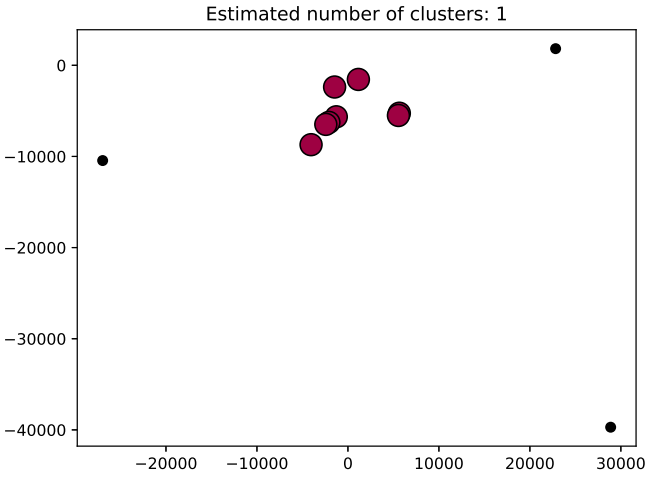
(p) Hour 15:00



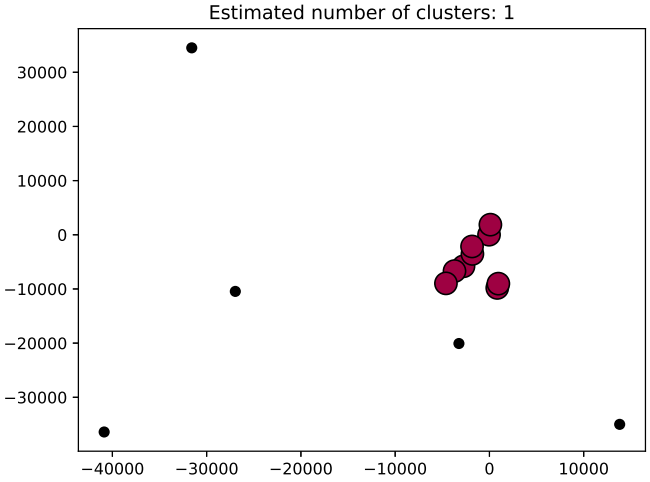
(q) Hour 16:00



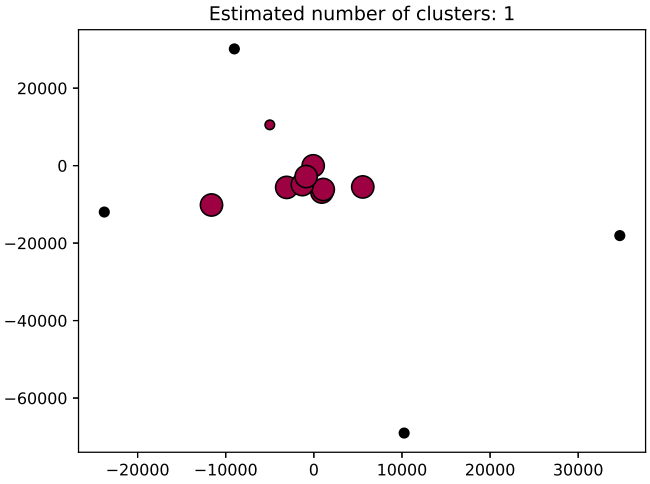
(r) Hour 17:00



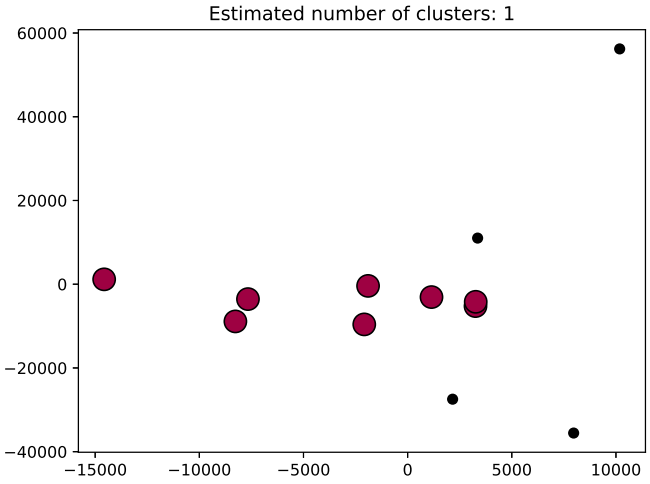
(s) Hour 18:00



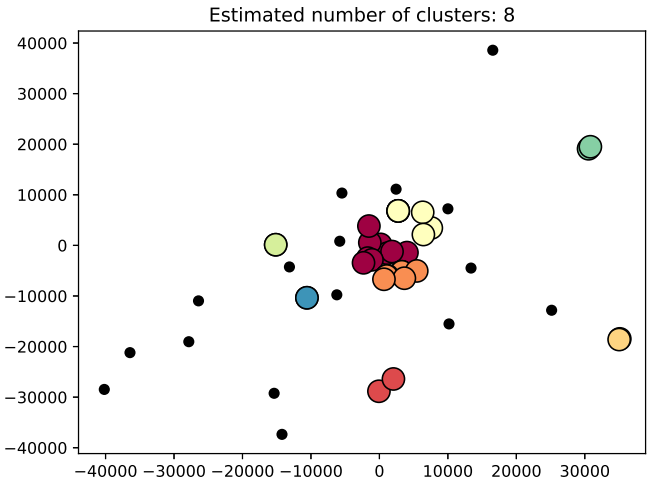
(t) Hour 19:00



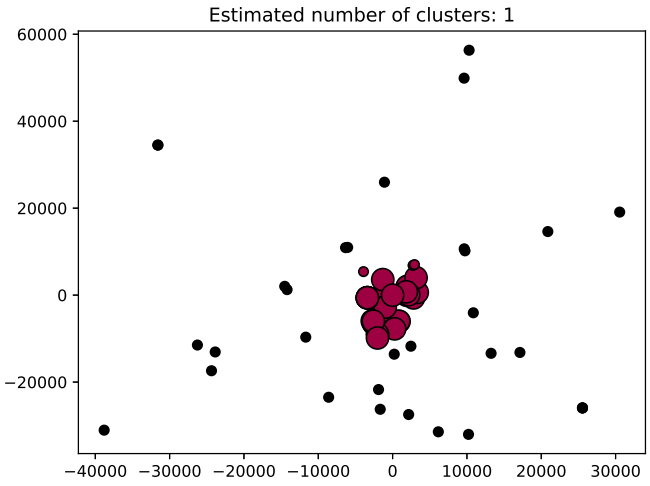
(u) Hour 20:00



(v) Hour 21:00



(w) Hour 22:00



(x) Hour 23:00

Figure 4. Clustering results for each hour.

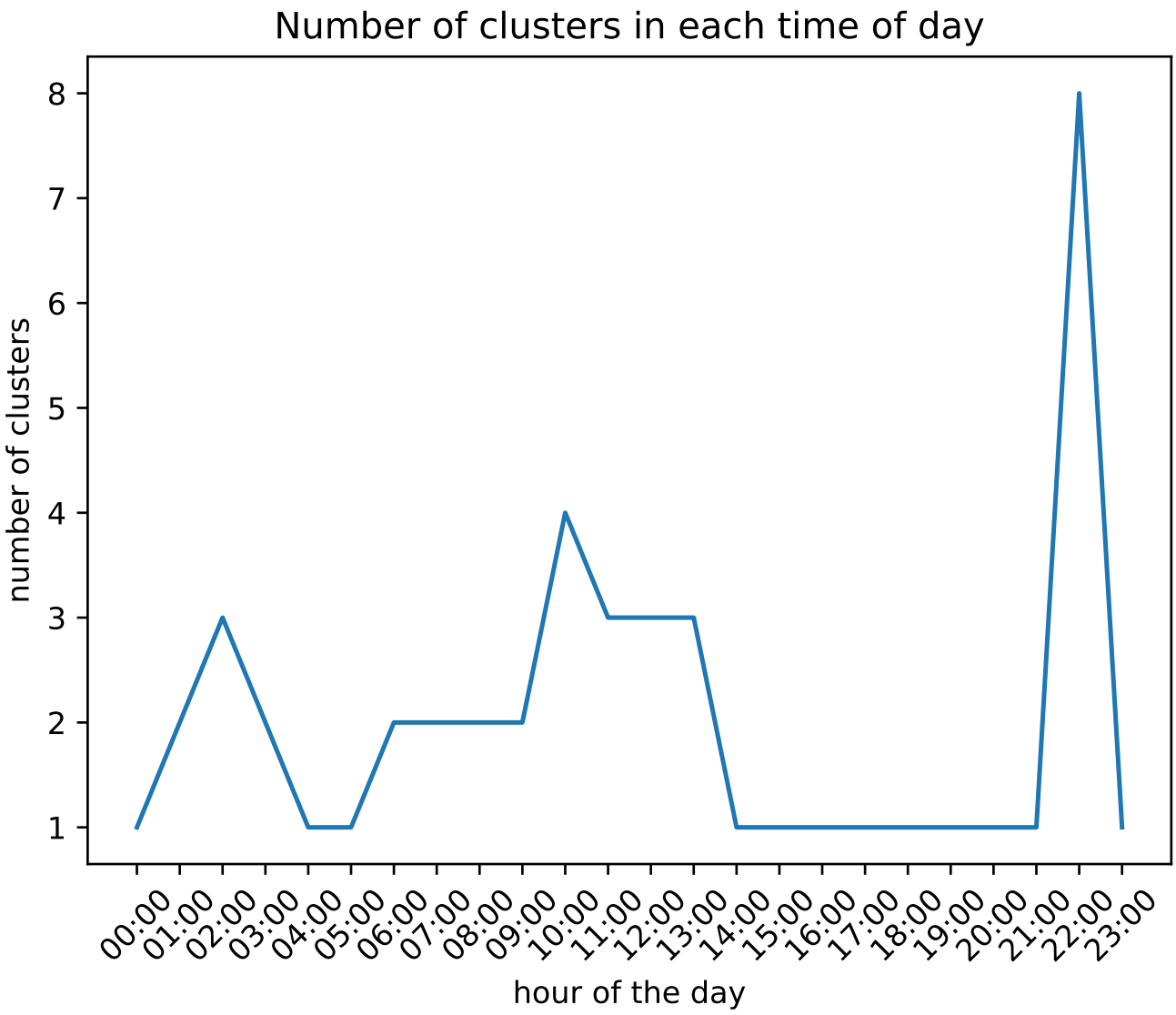


Figure 5. Variation of number of clusters with each hour.