hw3-matrix-factorization

November 14, 2017

In [1]:
```
# Adapted from
# https://bugra.github.io/work/notes/2014-04-19/alternating-least-squares-m
# but note that their algorithm is wrong
# because it doesn't take into account
# the missing values in alternating least squares
# Below is the correct algorithm
import numpy as np
Q = np.array(
    [[0, 1, np.nan],
     [1, np.nan, 1],
     [np.nan, 1, 2]])
lambda_ = 0.001
n_factors = 1
m, n = Q.shape
n_iterations = 10000
W = 1-(np.isnan(Q))*1
Q = np.nan_to_num(Q)


X = 5 * np.random.rand(m, n_factors)
Y = 5 * np.random.rand(n_factors, n)

def get_error(Q, X, Y, W):
    return np.sum((W*(Q - np.dot(X, Y)))**2)
for ii in range(n_iterations):
    for a in range(m):
        YW = np.dot(Y,np.diag(W[a,:]))
        X[a,:] = np.linalg.solve(np.dot(YW, YW.T) + lambda_ * np.eye(n_fact
                                 np.dot(Y, Q[a,:].T)).T
    for i in range(n):
        WX = np.dot(np.diag(W[:,i]),X)
        Y[:,i] = np.linalg.solve(np.dot(WX.T, WX) + lambda_ * np.eye(n_fact
                                 np.dot(X.T, Q[:,i]))
    if ii % 1000 == 0:
        print(get_error(Q, X, Y, W))
Q_hat = np.dot(X, Y)
print('Error of rated movies: {}'.format(get_error(Q, X, Y, W)))
print(Q_hat)
```

```
        print(X)
        print(Y)
```

```
0.865834399098
0.634544226429
0.634543228272
0.634543101749
0.634543080147
0.634543076245
0.634543075532
0.634543075402
0.634543075378
0.634543075374
Error of rated movies: 0.634543075373
[[ 0.48704756  0.61089466  0.95497563]
 [ 0.61089466  0.76623376  1.1978081 ]
 [ 0.95497563  1.1978081   1.87246283]]
[[ 0.69788791]
 [ 0.87534776]
 [ 1.36837959]]
[[ 0.69788795  0.87534782  1.36837968]]
```

```
In [3]: from numpy.linalg import norm
        print(X/norm(X))
        print(Y/norm(Y))
```

```
[[ 0.39481348]
 [ 0.49510937]
 [ 0.77394381]]
[[ 0.39481348  0.49510937  0.77394381]]
```

```
In [4]: def costgrad(u):
            cost = (u[0]*u[0])**2+2*(u[0]*u[1]-1)**2+2*(u[1]*u[2]-1)**2+(u[2]*u[2]-
            grad = np.array([4*u[0]**3+2*(u[0]*u[1]-1)*u[1],
                             2*(u[0]*u[1]-1)*u[0]+2*(u[1]*u[2]-1)*u[2],
                             2*(u[1]*u[2]-1)*u[1]+4*(u[2]*u[2]-2)*u[2]])
            #print(cost)
            return cost, grad

        import numpy as np
        from scipy.optimize import fmin_l_bfgs_b as minimize
        x = np.random.randn(3)
        optx,cost,messages = minimize(costgrad,x,epsilon=1e-18)

        print(optx)
        print(cost)
        print(optx/norm(optx))
```

```python
        print(costgrad(optx))
        #somehow the accuracy is quite bad but I'm not sure why
        #I suspect there are many local minimas
```

```
[-0.60237458 -0.87158359 -1.38758467]
0.676136093597
[-0.34503724 -0.49923886 -0.79480178]
(0.67613609359748583, array([-0.04632894, -0.00887743,  0.0490918 ]))
```

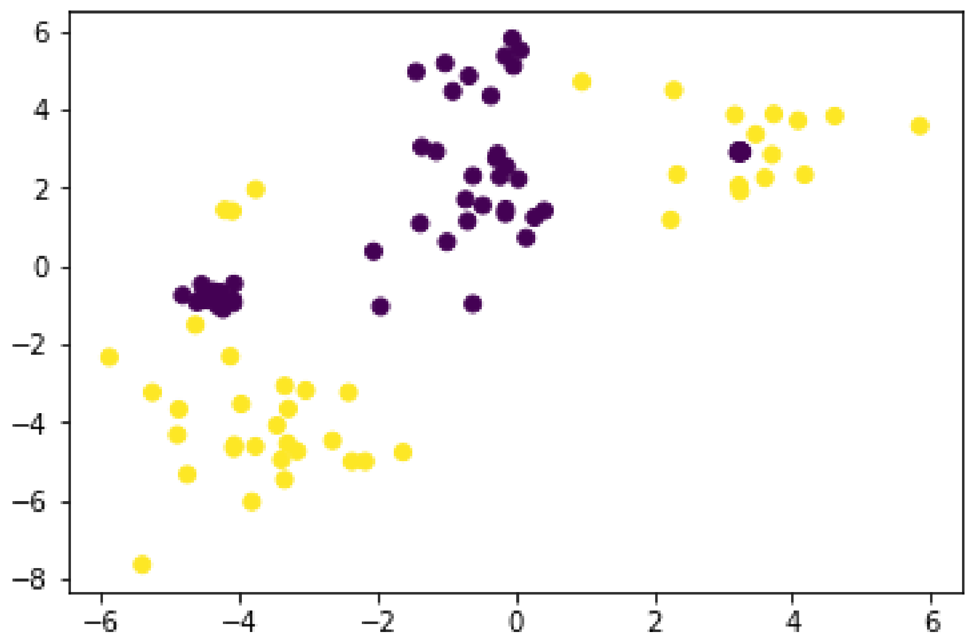
In [ ]:

In [ ]:

hw3-kernel-svm

October 20, 2017

In [8]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
csv = 'https://www.dropbox.com/s/wt45tvn9ig3o7vu/kernel.csv?dl=1'
data = np.genfromtxt(csv,delimiter=',')
X = data[:,1:]
Y = data[:,0]
plt.scatter(X[:,0],X[:,1],c=Y)
plt.show()
```

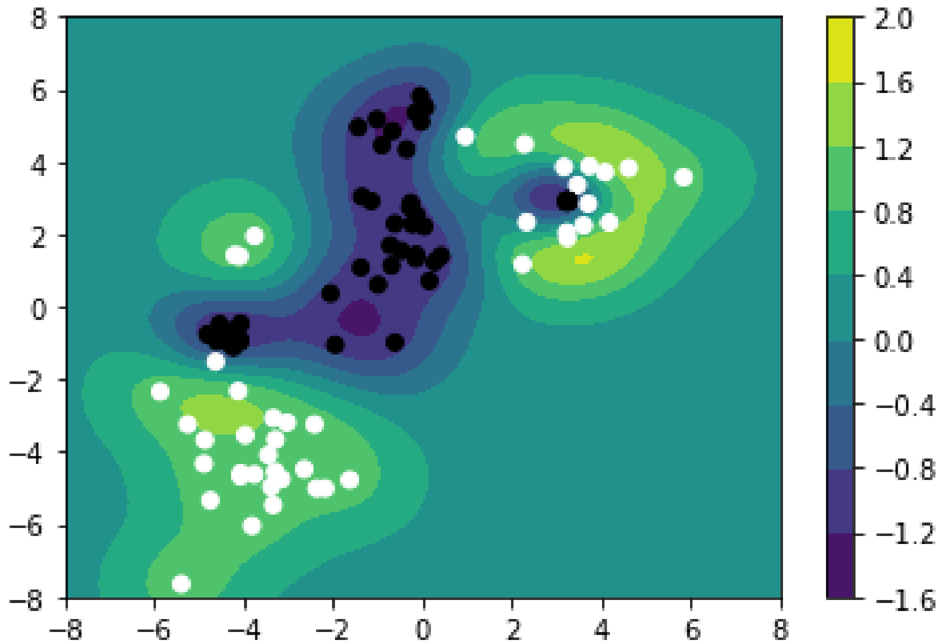


In [12]:
```
from sklearn import svm
clf = svm.SVC(gamma=0.5,kernel='rbf')
clf.fit(X, Y)

def decision(x1,x2,clf):
```

```
    return clf.decision_function(np.array([[x1,x2]]))[0]

vdecision = np.vectorize(decision,excluded=[2])
x1list = np.linspace(-8.0,8.0,100)
x2list = np.linspace(-8.0,8.0,100)
X1, X2 = np.meshgrid(x1list,x2list)
Z = vdecision(X1,X2,clf)
cp = plt.contourf(X1,X2,Z)
plt.colorbar(cp)
plt.scatter(X[:,0],X[:,1],c=Y,cmap='gray')
plt.show()
```



In [ ]:

hw3-deep-learning

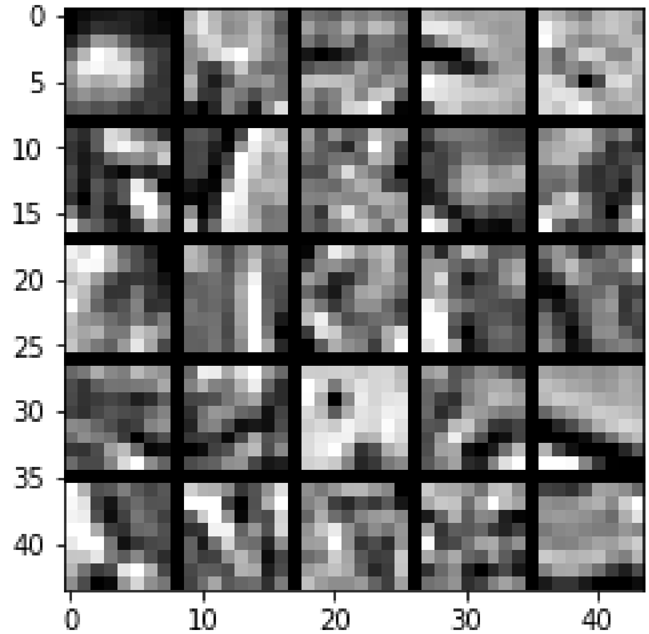
October 20, 2017

In [1]:
```
%matplotlib inline
import numpy as np
from numpy.linalg import norm
import matplotlib.pyplot as plt
from scipy.optimize import fmin_l_bfgs_b as minimize

from utils import normalize, tile_raster_images, sigmoid
from utils import ravelParameters, unravelParameters
from utils import initializeParameters
from utils import computeNumericalGradient

nV = 8*8        # number of visible units
nH = 25         # number of hidden units
dW = 0.0001     # weight decay term
sW = 3          # sparsity penalty term

npy = 'images.npy'
X = normalize(np.load(npy))
plt.imshow(tile_raster_images(X=X,
    img_shape=(8,8),tile_shape=(5,5),
    tile_spacing=(1,1)),cmap='gray')
plt.show()
```



In [2]:
```
def sparseAutoencoderCost(theta,nV,nH,dW,sW,X):

    W1,W2,b1,b2 = unravelParameters(theta,nH,nV)
    n = X.shape[0]

    z2 = np.dot(X,W1)+np.dot(np.ones((n,1)),b1.T)
    a2 = sigmoid(z2)
    z3 = np.dot(a2,W2)+np.dot(np.ones((n,1)),b2.T)
    a3 = sigmoid(z3)

    eps = a3-X
    loss = 0.5*np.sum(eps**2)/n
    decay = 0.5*(np.sum(W1**2)+np.sum(W2**2))

    rho = 0.01
    a2mean = np.mean(a2,axis=0).reshape(nH,1)
    kl = np.sum(rho*np.log(rho/a2mean)+\
         (1-rho)*np.log((1-rho)/(1-a2mean)))
    dkl = -rho/a2mean+(1-rho)/(1-a2mean)
    cost = loss+dW*decay+sW*kl

    d3 = eps*a3*(1-a3)
    d2 = (sW*dkl.T+np.dot(d3,W2.T))*a2*(1-a2)
    W1grad = np.dot(X.T,d2)/n+dW*W1
    W2grad = np.dot(a2.T,d3)/n+dW*W2
    b1grad = np.dot(d2.T,np.ones((n,1)))/n
```

```
        b2grad = np.dot(d3.T,np.ones((n,1)))/n

        grad = ravelParameters(W1grad,W2grad,b1grad,b2grad)
        print(' .',end="")
        return cost,grad
```

In [3]:
```
# Obtain random parameters theta
theta = initializeParameters(nH,nV)
cost,grad = sparseAutoencoderCost(theta,nV,nH,dW,sW,X)
```

```
 .
```

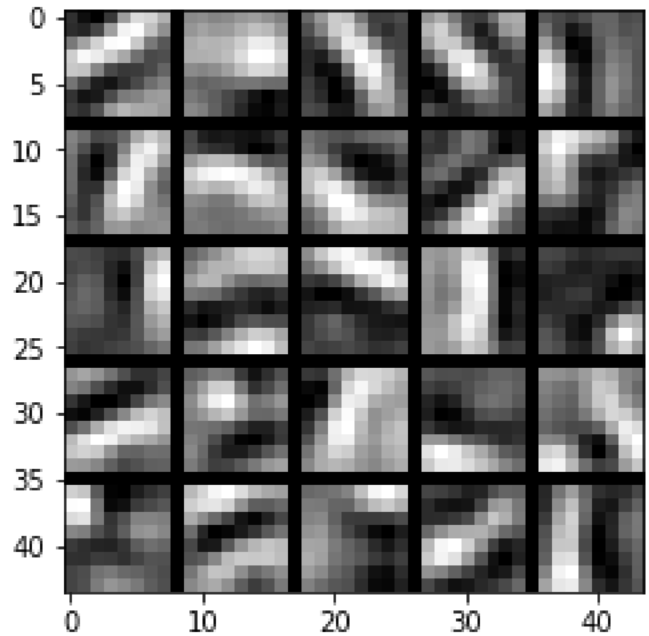
In [4]:
```
print('\nComparing numerical gradient with backprop gradient')
num_coords = 5
indices = np.random.choice(theta.size,num_coords,replace=False)
numgrad = computeNumericalGradient(lambda t:
    sparseAutoencoderCost(t,nV,nH,dW,sW,X)[0],theta,indices)
subnumgrad = numgrad[indices]
subgrad = grad[indices]
diff = norm(subnumgrad-subgrad)/norm(subnumgrad+subgrad)
print('\n',np.array([subnumgrad,subgrad]).T)
print('The relative difference is',diff)
```

```
Comparing numerical gradient with backprop gradient
 . . . . . . . . . .
 [[ 1.0850302   1.0850302 ]
 [-0.0114521  -0.0114521 ]
 [-0.00444169 -0.00444169]
 [ 0.01398969  0.01398969]
 [ 0.90842197  0.90842197]]
The relative difference is 4.61269902073e-11
```

In [5]:
```
print('\nTraining neural network')
theta = initializeParameters(nH,nV)
opttheta,cost,messages = minimize(sparseAutoencoderCost,
    theta,fprime=None,maxiter=400,args=(nV,nH,dW,sW,X))

W1,W2,b1,b2 = unravelParameters(opttheta,nH,nV)
plt.imshow(tile_raster_images(X=W1.T,
    img_shape=(8,8),tile_shape=(5,5),
    tile_spacing=(1,1)),cmap='gray')
plt.show()
```

```
Training neural network
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```



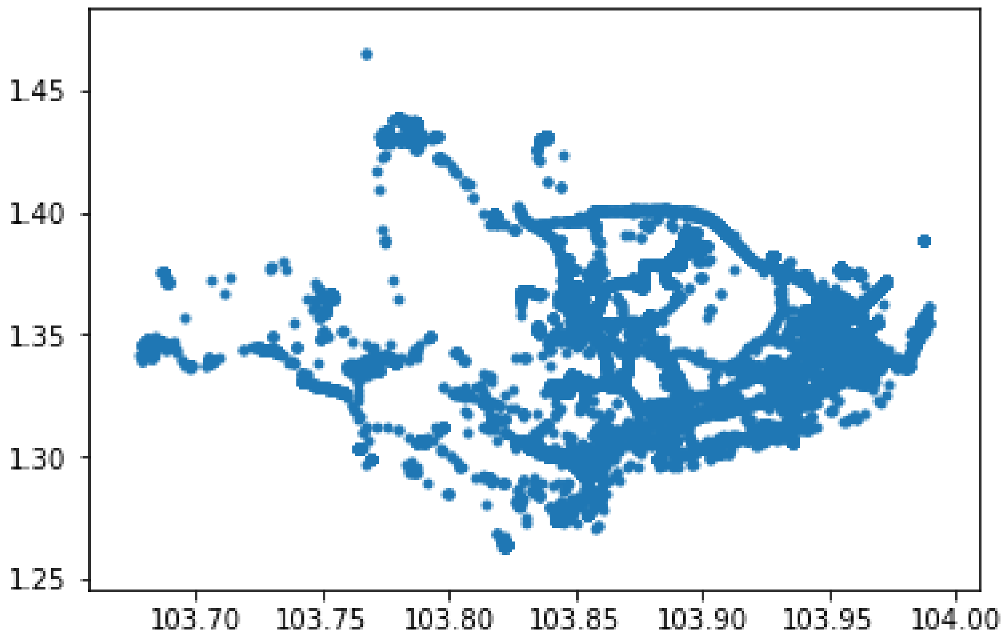

```
In [ ]:
```

hw3-dataspark

October 20, 2017

```
In [260]: # hint large dataset, so work on sample first
          %matplotlib inline
          import numpy as np
          import pandas as pd
          import matplotlib.pyplot as plt
          from IPython.display import display
          data = pd.read_csv('dataspark.csv')
          #data.loc[:,'user'] = data['userid'].astype('category').cat.codes
          #data = data.drop(['seqid','userid','index','acc','dir','spd'],axis=1)
          data = data.drop(['seqid','index','acc','dir','spd'],axis=1)
          print(data.info())
          plt.scatter(data['lon'],data['lat'],marker='.')
          plt.show()
```

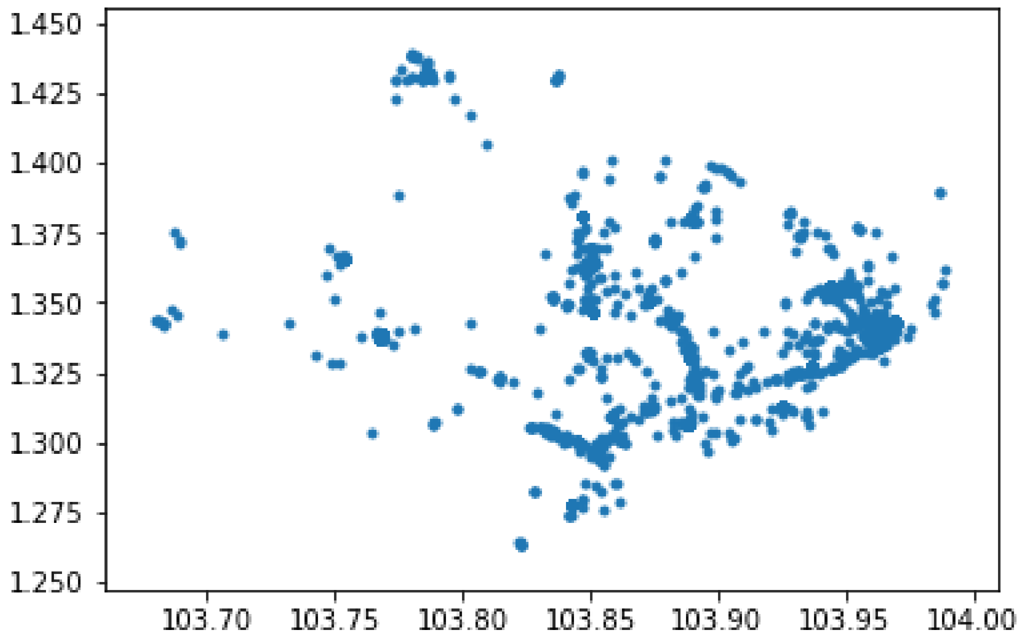
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 141437 entries, 0 to 141436
Data columns (total 4 columns):
date      141437 non-null object
userid    141437 non-null object
lat       141437 non-null float64
lon       141437 non-null float64
dtypes: float64(2), object(2)
memory usage: 4.3+ MB
None
```



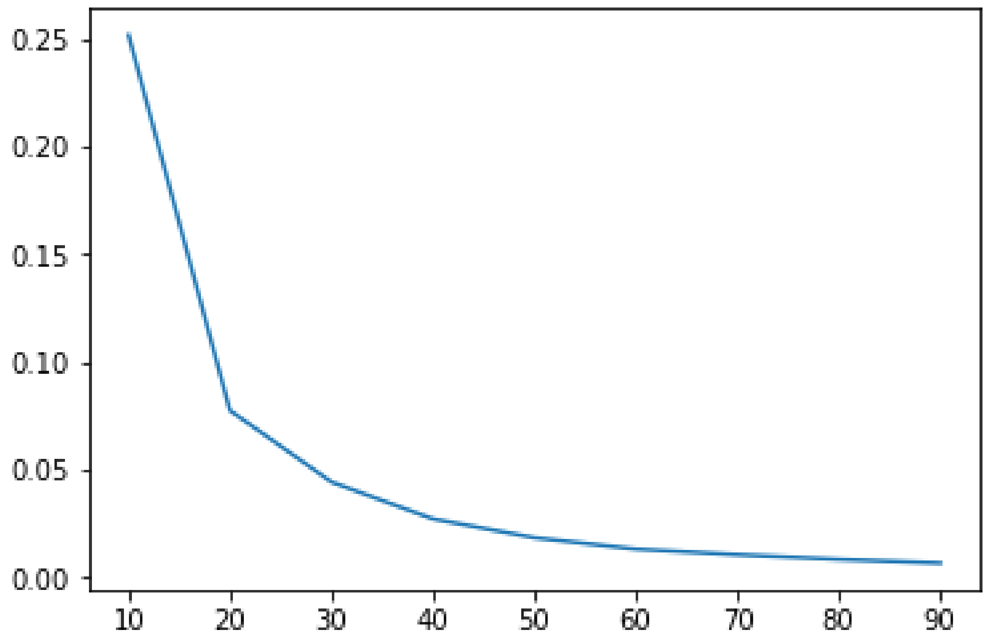

```
In [261]: #data = data.sample(frac=0.05,random_state=200)
```

```
In [262]: data['date'] = pd.DatetimeIndex(data['date']).round('5min')
          data = data.groupby(['userid','date']).mean().reset_index()
          print(data.info())
          plt.scatter(data['lon'],data['lat'],marker='.')
          plt.show()
```

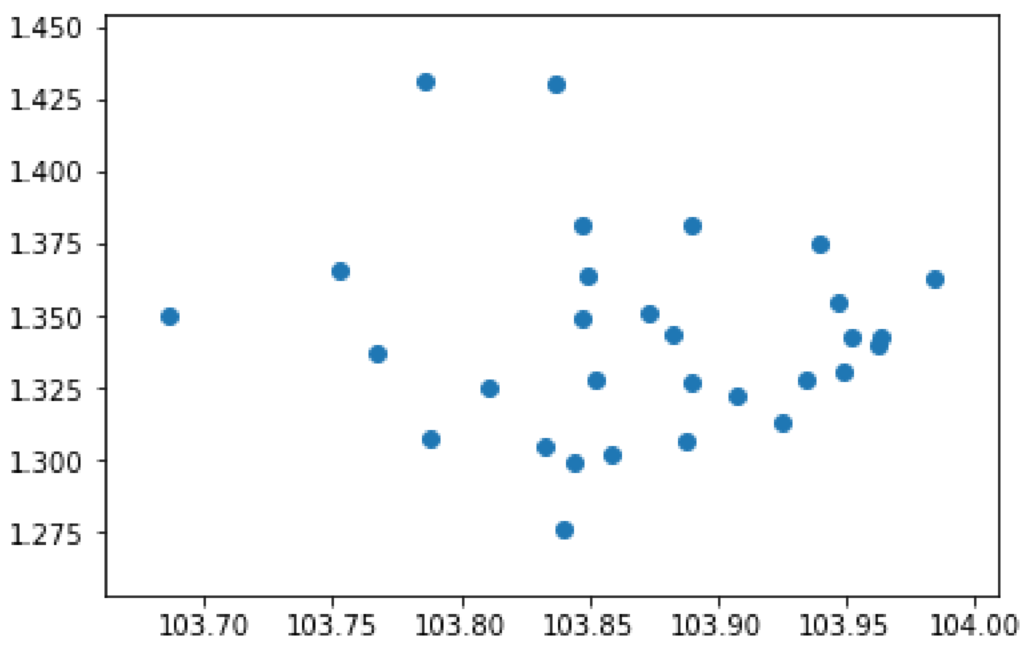
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4950 entries, 0 to 4949
Data columns (total 4 columns):
userid    4950 non-null object
date      4950 non-null datetime64[ns]
lat       4950 non-null float64
lon       4950 non-null float64
dtypes: datetime64[ns](1), float64(2), object(1)
memory usage: 154.8+ KB
None
```




```
In [285]: from sklearn.cluster import KMeans
          smp = data[['lat','lon']].sample(n=3000,random_state=200)
          score = []
          cls_range = list(range(10,100,10))
          for num_cls in cls_range:
              kmeans = KMeans(n_clusters=num_cls,random_state=0).fit(smp)
              score = np.append(score,[kmeans.inertia_])
          plt.plot(cls_range,score)
          plt.show()
```




```
In [287]: num_clusters = 30
          kmeans = KMeans(n_clusters=num_clusters,random_state=0).fit(data[['lat','
          centroids = kmeans.cluster_centers_
          plt.scatter(centroids[:,1],centroids[:,0])
          plt.show()
          centroids
```

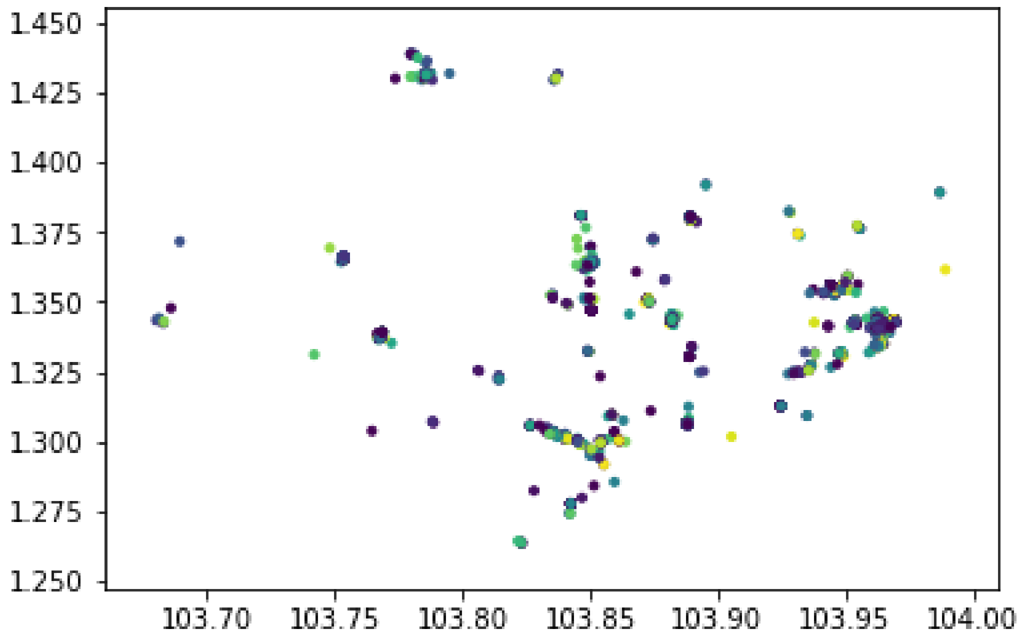


```
Out[287]: array([[   1.34231774,  103.96394285],
                 [   1.33745237,  103.76749434],
                 [   1.35147658,  103.87297833],
                 [   1.43124535,  103.7854103 ],
                 [   1.27631415,  103.84037286],
                 [   1.30664982,  103.88805404],
                 [   1.35474626,  103.94663198],
                 [   1.36387751,  103.84971366],
                 [   1.32783532,  103.93423465],
                 [   1.35023557,  103.68595643],
                 [   1.38185233,  103.88942686],
                 [   1.36545601,  103.75301077],
                 [   1.30219691,  103.85859149],
                 [   1.34355461,  103.88215168],
                 [   1.34902368,  103.84722035],
                 [   1.4303255 ,  103.83676323],
                 [   1.304894  ,  103.83237436],
                 [   1.30801309,  103.78777959],
                 [   1.32239796,  103.90747823],
                 [   1.38182927,  103.84754873],
                 [   1.32741138,  103.88966824],
                 [   1.31327842,  103.92557858],
                 [   1.33045728,  103.94852412],
                 [   1.37550194,  103.93941288],
                 [   1.32844829,  103.85284033],
                 [   1.36312499,  103.98426448],
                 [   1.34004076,  103.96246783],
                 [   1.34297374,  103.95232024],
                 [   1.32511868,  103.81083421],
                 [   1.29977625,  103.84455839]])
```

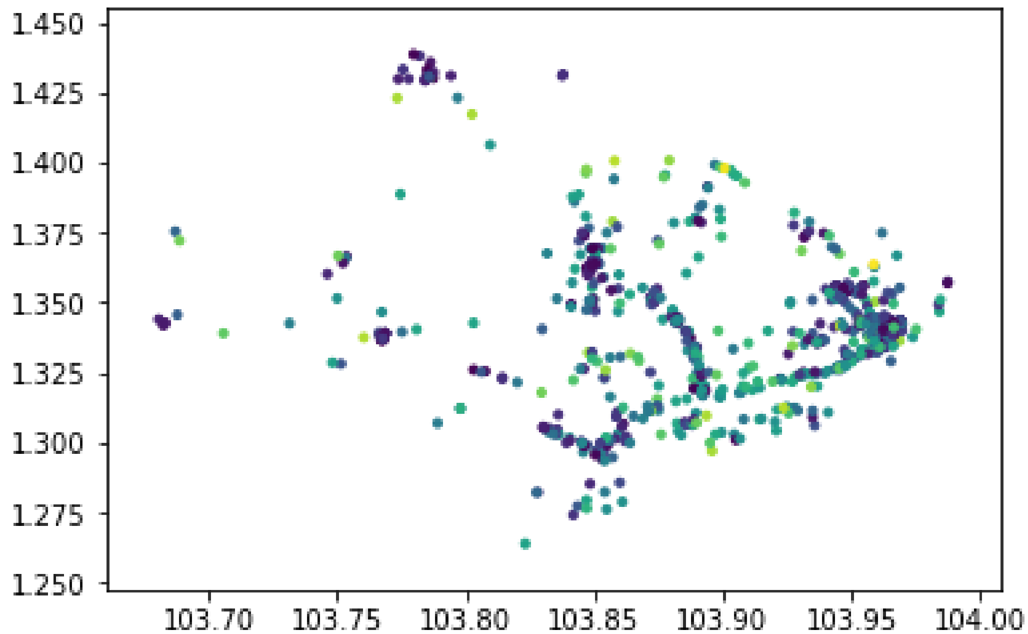
```
In [283]: from numpy.linalg import norm
          for u in data['userid'].unique():
              user = data[data['userid']==u]
              date = pd.DatetimeIndex(user['date'])
              hour = (date-date[0])/np.timedelta64(1,'h')
              latlon = user[['lat','lon']].get_values()
              index = range(user.shape[0]-1)
              time = np.array([hour[x+1]-hour[x] for x in index])
              dist = np.array([norm(latlon[x+1]-latlon[x]) for x in index])
              speed = dist/time*111
              speed = np.append(speed,[0])
              data.loc[data['userid']==u,'speed'] = speed
```

```
In [284]: stop = data[data['speed']<1]
          plt.scatter(stop['lon'],stop['lat'],c=np.log(stop['speed']+1),marker='.')
```

```python
plt.show()
print('number of entries =',stop.shape[0])
move = data[data['speed']>1]
plt.scatter(move['lon'],move['lat'],c=np.log(move['speed']+1),marker='.')
plt.show()
print('number of entries =',move.shape[0])
```

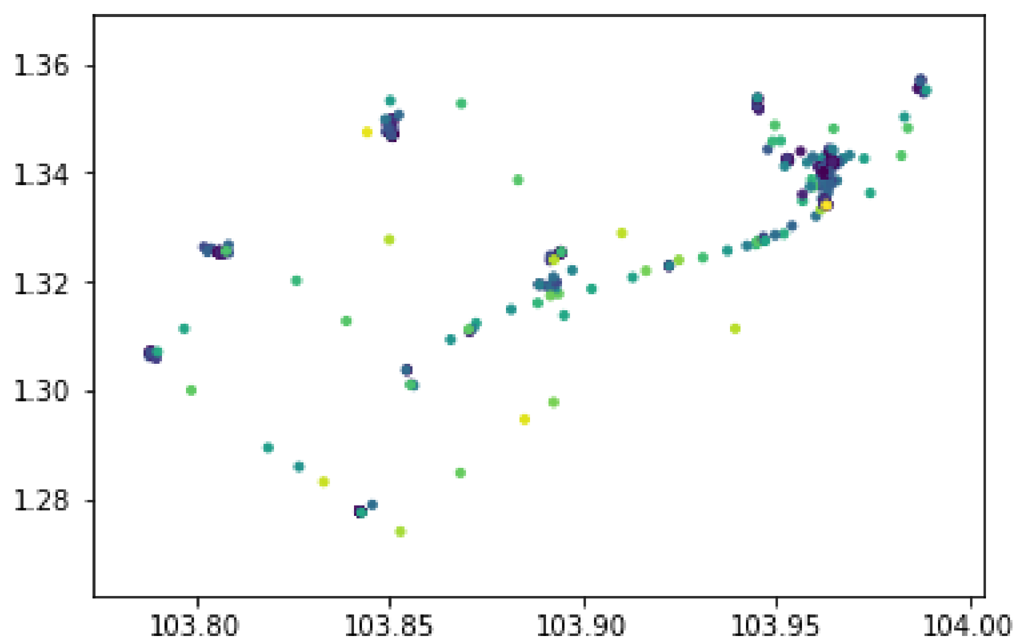



```
number of entries = 4028
```




```
number of entries = 922
```

```
In [228]: #m = data[data['user']==np.random.randint(data['user'].max()+1)]
          #plt.scatter(m['lon'],m['lat'],c=np.log(m['speed']+1),marker='.')
          #plt.show()
```



In [225]:
```
#get api key
#https://github.com/pbugnion/gmaps
#conda install -c conda-forge gmaps
#start a fresh ipynb
import gmaps
import gmaps.datasets
gmaps.configure(api_key="AIzaSyDemoP0bxfvyo2CG0C3q7UIKXnlzKHKauA")
locations = gmaps.datasets.load_dataset("taxi_rides")
fig = gmaps.figure()
fig.add_layer(gmaps.heatmap_layer(locations))
fig
```

```
Figure()
```

In [288]:
```
fig = gmaps.figure()
fig.add_layer(gmaps.heatmap_layer(data[['lat','lon']]))
fig
```

```
Figure()
```

In [290]:
```
fig = gmaps.figure()
fig.add_layer(gmaps.heatmap_layer(centroids))
fig
```

```
Figure()
```

In [ ]: