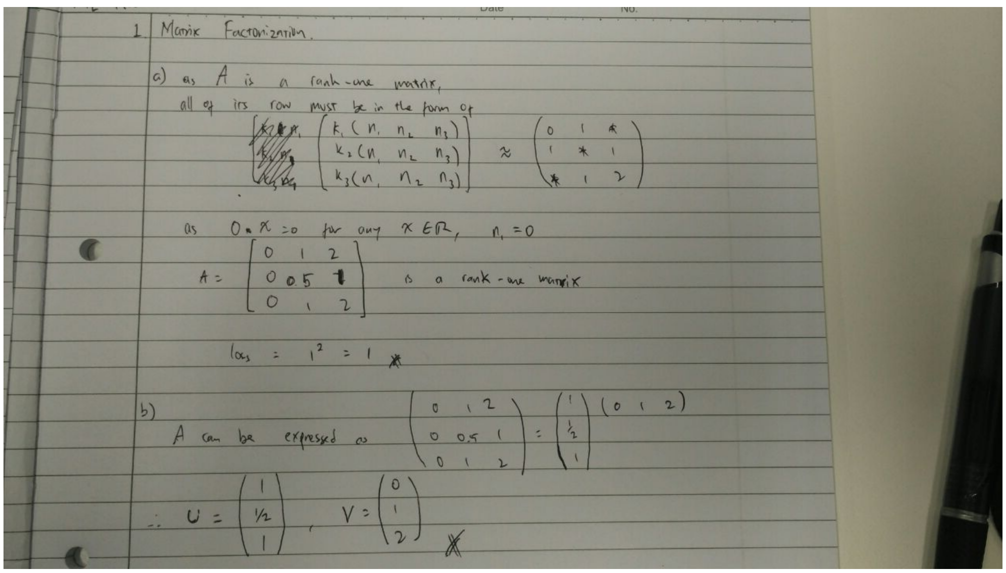
hw3

October 12, 2017

1 ML Homework 3

Gede Ria Ghosalya - 1001841

1.0.1 Q1. Matrix Factorization


1. Matrix Factorization.

a) as A is a rank-one matrix, all of its row must be in the form of

$$\begin{bmatrix} k_1(n_1 & n_2 & n_3) \\ k_2(n_1 & n_2 & n_3) \\ k_3(n_1 & n_2 & n_3) \end{bmatrix} \approx \begin{pmatrix} 0 & 1 & * \\ 1 & * & 1 \\ * & 1 & 2 \end{pmatrix}$$

as $0 \cdot x = 0$ for any $x \in \mathbb{R}$, $n_1 = 0$

$$A = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 0.5 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$ is a rank-one matrix

$loss = 1^2 = 1$

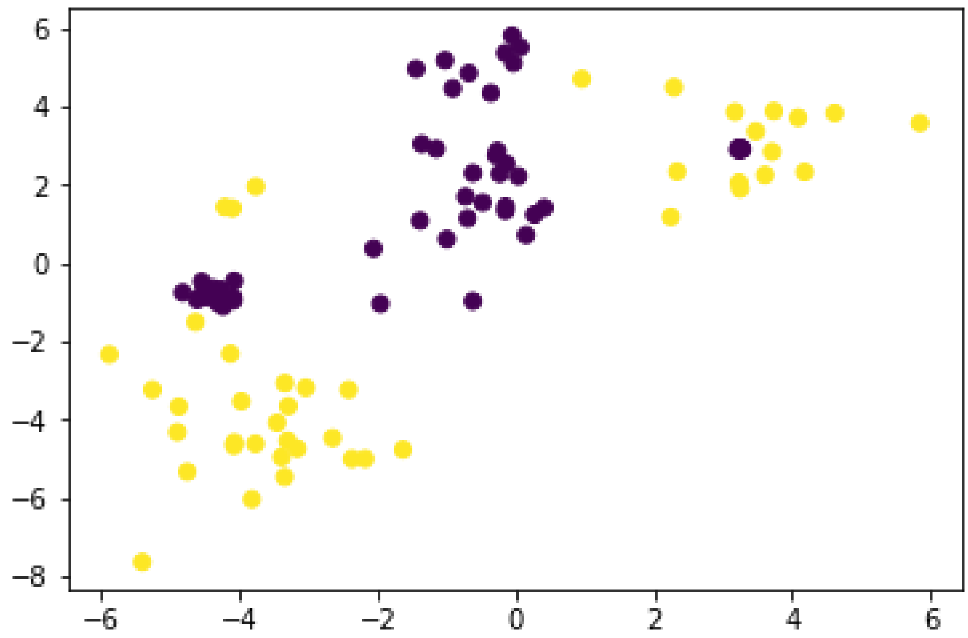
b) A can be expressed as $\begin{pmatrix} 0 & 1 & 2 \\ 0 & 0.5 & 1 \\ 0 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 2 \end{pmatrix}$

$\therefore U = \begin{pmatrix} 1 \\ \frac{1}{2} \\ 1 \end{pmatrix}$, $V = \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}$

pic

1.0.2 Q2. Support Vector Machines

```
In [1]: %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()
```



---

a) Use the `sklearn.svm.SVC` module to train a kernel support vector machine via the radial basis kernel. Set gamma to 0.5 and `kernel` to `rbf`.

In [2]:
```
from sklearn.svm import SVC
dis_svc = SVC(gamma=0.5, kernel='rbf')
dis_svc.fit(X, Y)
```

Out[2]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.5, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

---

b) Evaluate the kernel SVM's decision function. You may use the `decision_function` method in `SVC`. Write a function that takes coordinates `x1, x2` and the SVC object `clf`, and return the value of decision function.

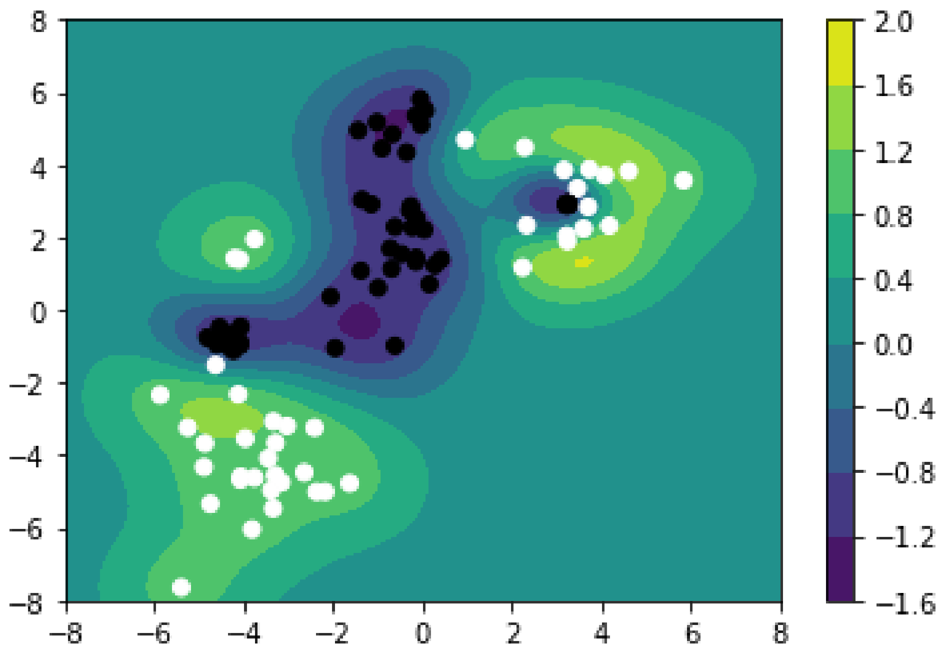
In [3]:
```
clf = dis_svc

def decision(x1, x2, clf):
    x = np.array([[x1, x2]])
    val = clf.decision_function(x)
    return val[0]

decision(3,5,clf)
```

```
Out[3]: 1.1813851778172231
```

---

c) Use the following code to visualize the classifier and the data points.

```
In [4]: 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()
```



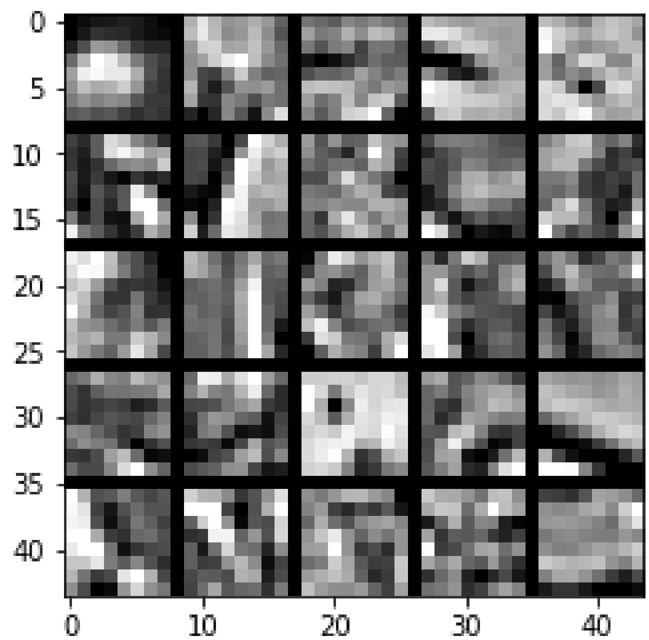
1.0.3 Q3. Deep Learning

```
In [5]: %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
```

In [6]:
```
nV = 8*8
nH = 25
dW = 0.0001
sW = 3
```

In [7]:
```
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()
```



---

a) We implement the function which computes the cost and the gradient of the sparse autoencoder. This function will be passed to an optimization engine, together with the `theta` vector that contains the current state of all the model parameters. The first step of the function is therefore to unpack the `theta` vector into *W1, W2, b1, b2*.

In [8]:
```
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 = norm(eps)**2/(2*n)
        decay = 0.5*(norm(W1)**2+norm(W2)**2)

        #compute sparsity terms and total cost
        rho = 0.01
        a2mean = np.mean(a2, axis=0).reshape(nH, 1)
        kl_first = rho*np.log(rho/a2mean)
        kl_last = (1-rho)*np.log((1-rho)/(1-a2mean))
        kl = np.sum(kl_first + kl_last)
        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 [9]:
```
theta = initializeParameters(nH, nV)
cost, grad = sparseAutoencoderCost(theta, nV, nH, dW, sW, X)
print(cost,grad)
#print(np.ones(50))
```

```
 .56.5319053844 [ 0.83106574  0.87429827  0.80847634 ..., -0.02399919 -0.03810212
 -0.00833089]
```

---

b) Compare the backdrop gradient in `sparseAutoencoderCost` with the gradient computed numerically from the cost. The relative difference should be less than 10e-9.

In [10]:
```
print('\nComparing numerical gradient with backdrop 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('Relative difference: ', diff)

        if diff < 10**(-9):
            print("small enough!")
        else:
            print("NOOOOO!!")
```

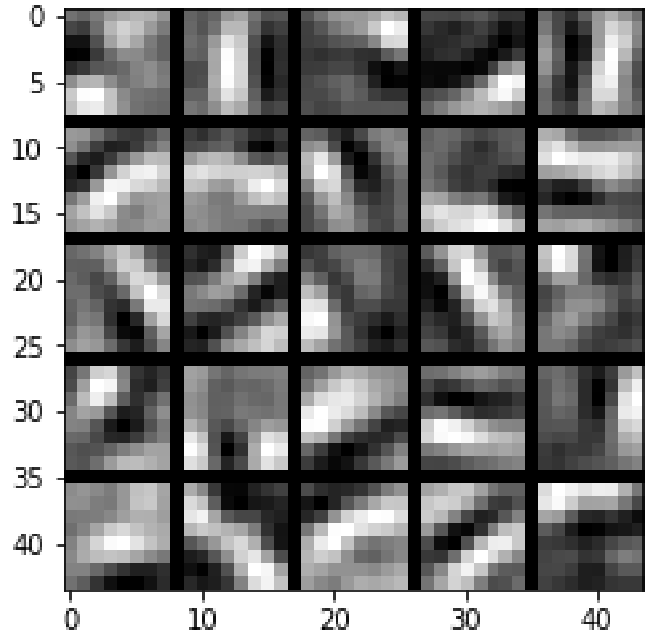
```
Comparing numerical gradient with backdrop gradient
 . . . . . . . . . .
 [[ -5.60882391e-03  -5.60882395e-03]
 [  8.31224652e-01   8.31224652e-01]
 [ -4.60600091e-05  -4.60599727e-05]
 [  1.60391841e-02   1.60391841e-02]
 [  5.93707750e-01   5.93707750e-01]]
Relative difference:  9.6956630679e-11
small enough!
```

---

c) Finally, run the following code to train the deep neural network and to visualize the features learnt by the autoencoder. The optimization takes several minutes.

```
In [11]: print('\nTraining neural network')
         theta = initializeParameters(nH,nV)
         opttheta,cost,msg = 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
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```



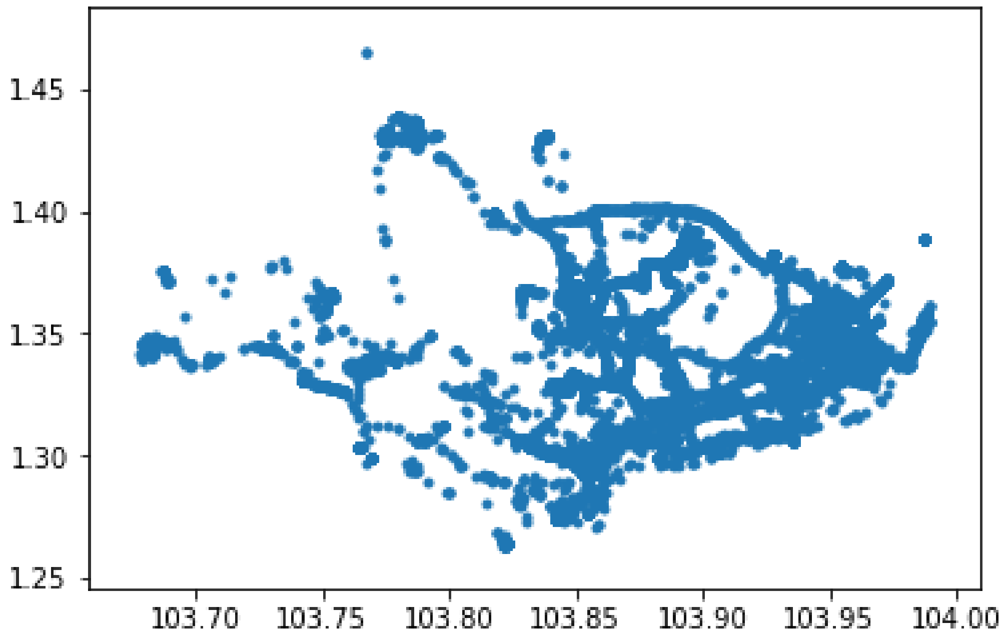
1.0.4 Q4. DataSpark Challenge

```
In [12]: %matplotlib inline
         import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         from IPython.display import display
```

```
In [13]: data = pd.read_csv('dataspark.csv')
         data = data.drop(['seqid','index','acc','dir','spd'],
                          axis=1)
         print(data.info())
         plt.scatter(data['lon'],data['lat'],marker='.')
         plt.show()
         #data = data.sample(frac=0.05, random_state=200)
```

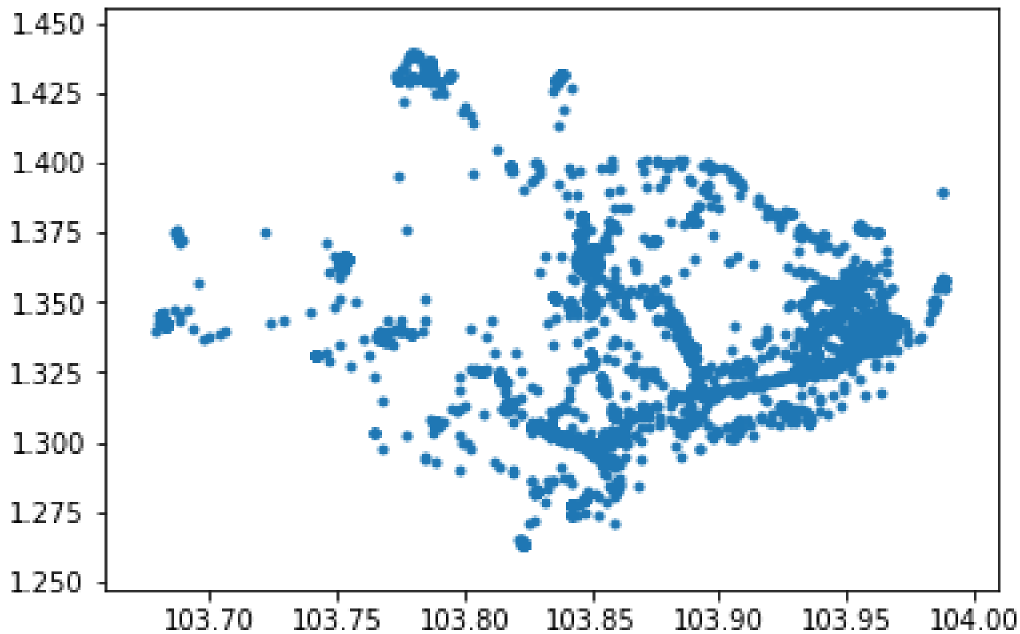
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 141437 entries, 0 to 141436
Data columns (total 4 columns):
date       141437 non-null object
userid     141437 non-null object
lat        141437 non-null float64
lon        141437 non-null float64
dtypes: float64(2), object(2)
memory usage: 4.3+ MB
```

```
None
```




```
In [14]: 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()
```

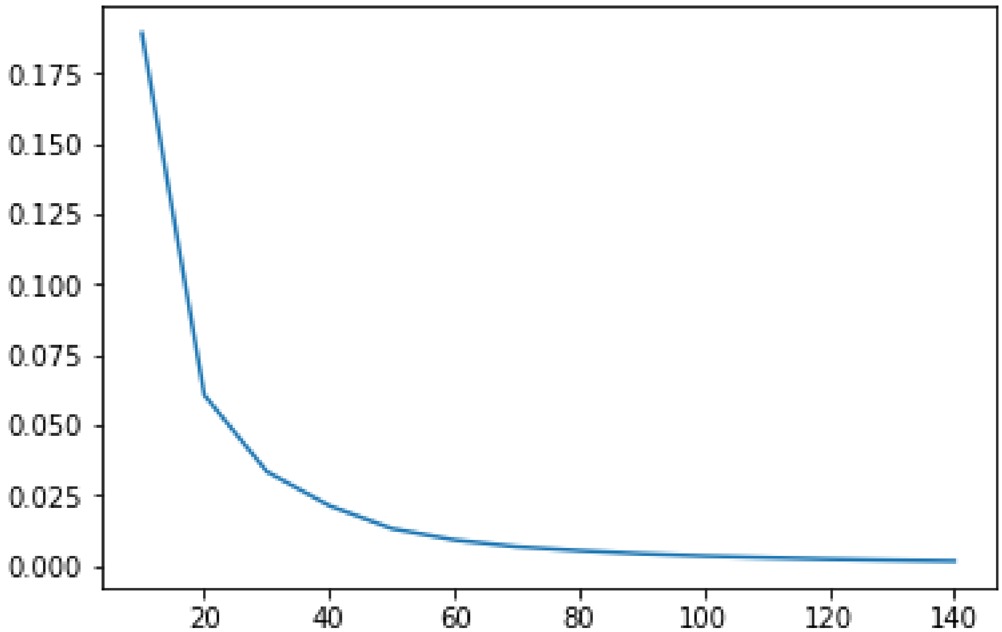
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26115 entries, 0 to 26114
Data columns (total 4 columns):
userid    26115 non-null object
date      26115 non-null datetime64[ns]
lat       26115 non-null float64
lon       26115 non-null float64
dtypes: datetime64[ns](1), float64(2), object(1)
memory usage: 816.2+ KB
None
```



---

a) Cluster the GPS locations for all users to find commonly visited places. Us the 'elbow' method to find a suitable number of clusters. Sample the data to improve speed. Write your guess for the number of clusters in the data.

```
In [15]: from sklearn.cluster import KMeans
         smp = data[['lat','lon']].sample(n=3000,random_state=200)
         score = []
         cls_range = list(range(10,150,10))
         for num_cls in cls_range:
             n_cluster = KMeans(n_clusters=num_cls).fit(smp)
             score.append(n_cluster.inertia_)
         plt.plot(cls_range,score)
         plt.show()
```

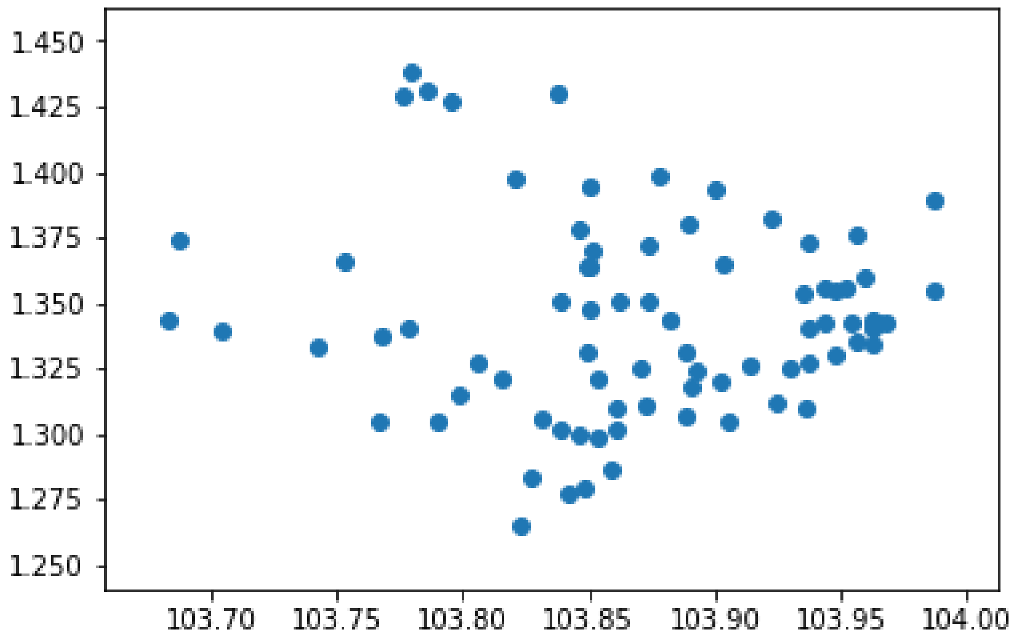


My guess is that we have 80 clusters.

---

b) Visualize the trained centroids. In the code below, `centroid` is a numpy array where each row consists of the latitude of some centroid.

```
In [16]: num_cluster = 80
         data_latlon = data[['lat','lon']]
         decided_kmean = KMeans(n_clusters=num_cluster).fit(data_latlon)
         centroids = decided_kmean.cluster_centers_
         plt.scatter(centroids[:,1],centroids[:,0])
         plt.show()
```

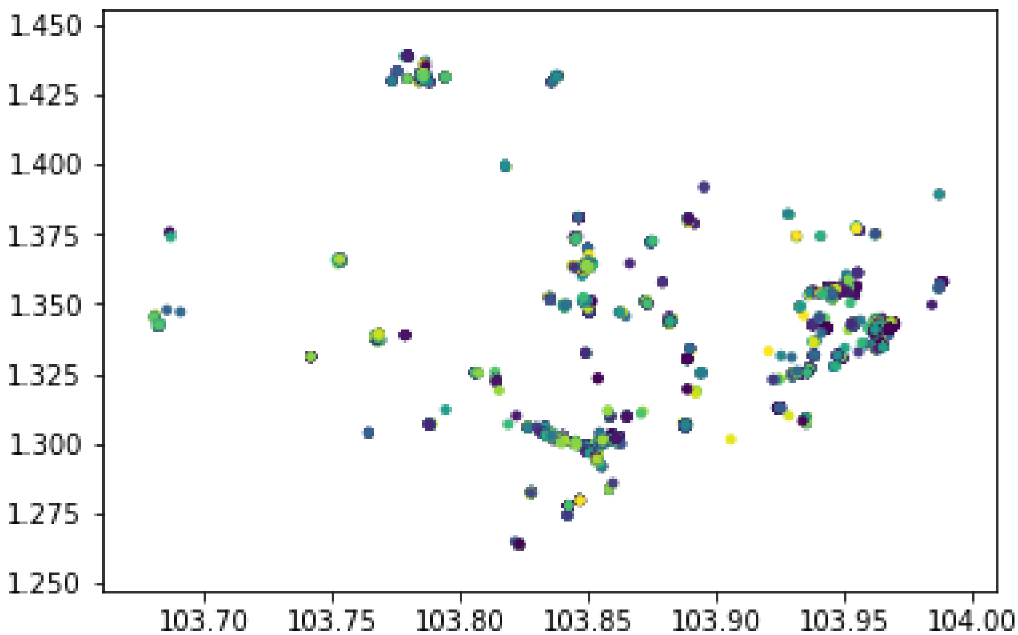


---

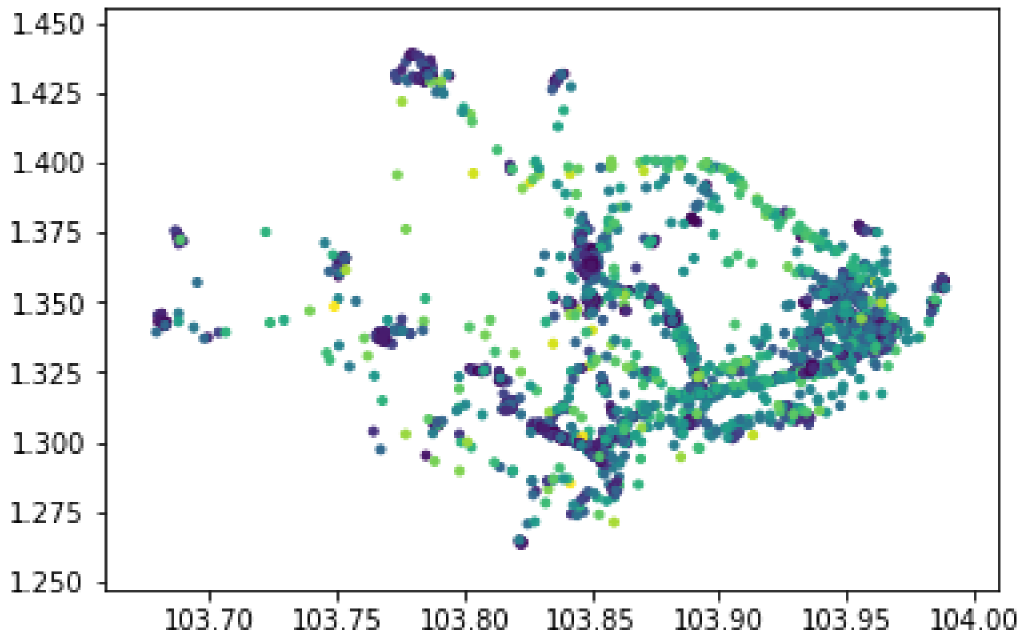
c) Compute the speeds at which each user is travelling.

```
In [47]: 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')
             hourfor = np.append(hour[1:], hour[-1])
             dur = hourfor - hour

             latlon = user[['lat','lon']].get_values()
             latlonfor = np.vstack([latlon[1:],latlon[-1]])
             displacement = (latlonfor - latlon)**2
             disp_p = (displacement[:,0] + displacement[:,1])**0.5

             speed = 111*disp_p/dur
             data.loc[data['userid']==u,'speed'] = speed

In [49]: stop = data[data['speed']<1]
         plt.scatter(stop['lon'],stop['lat'],
                     c=np.log(stop['speed']+1),
                     marker='.')
         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 = 20921
```




```
Number of entries = 5164
```