Clustering:

1. K-means
2. Fuzzy C-means
3. GMM
4. Practical Example (repeat for all 3)

1. K-means clustering

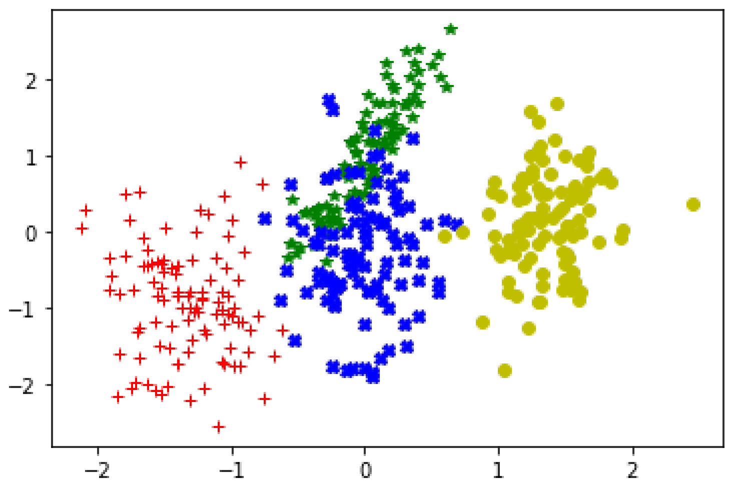
a) Data genearation

b) Generate 2D gaussian data of 4 types each having 100 points, by taking appropriate mean and varince (example: mean :(0.5 0) (5 5) (5 1) (10 1.5), variance : Identity matrix)

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt

## Data generation
# write your code here
```

```
(400,)
```



Cluster Initialization

a) Randomly initialize the cluster centers (any k- number of data points from the generated data)

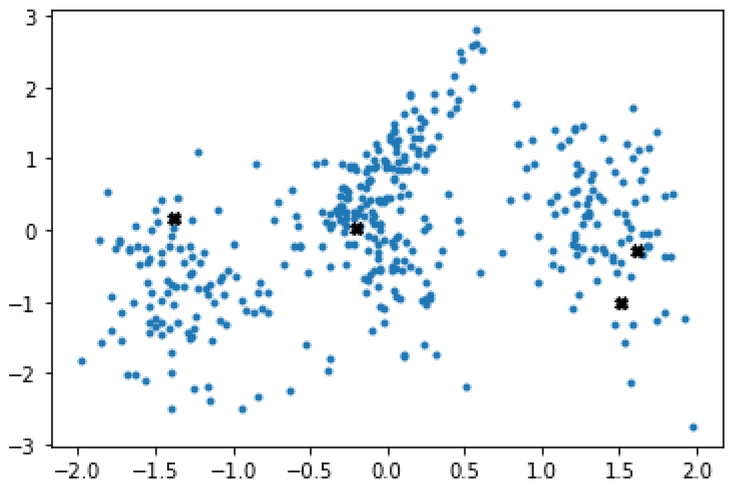
In [ ]:

```
# write your code here
```

```
[[ 1.60408577 -0.2851104 ]
 [ 1.50414077 -1.00625275]
 [-0.20228245  0.01791973]
 [-1.37804389  0.17145411]]
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f2c24a31eb8>]
```



Cluster assignment and re-estimation Stage

a) Using initial/estimated cluster centers (mean μ_i) perform cluster assignment.

b) Assigned cluster for each feature vector (X_j) can be written as:

$$arg \min_i ||C_i - X_j||_2, \; 1 \leq i \leq K, \; 1 \leq j \leq N$$

c) Re-estimation: After cluster assignment, the mean vector is recomputed as,

$$\mu_i = \frac{1}{N_i} \sum_{j \in i^{th} cluster} X_j$$

where N_i represents the number of datapoints in the i^{th} cluster.

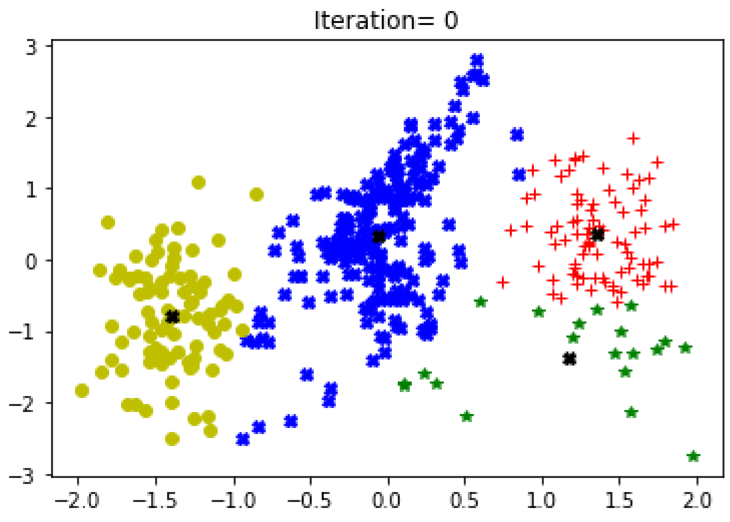
d) Objective function (to be minimized):

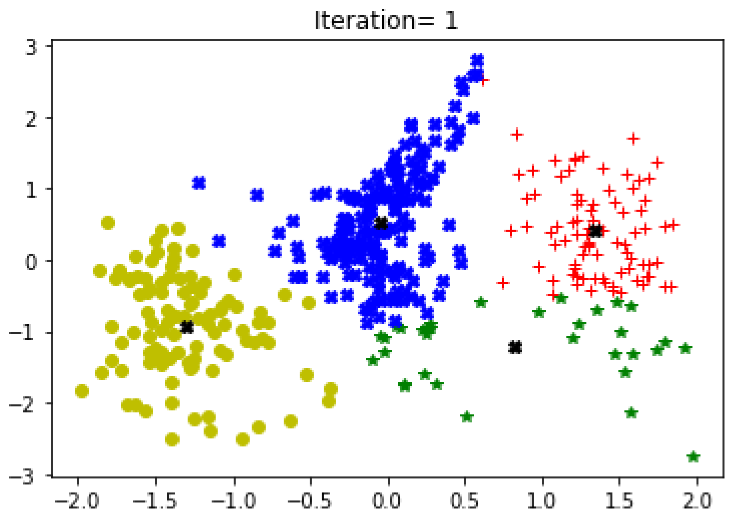
$$Error(\mu) = \frac{1}{N} \sum_{i=1}^{K} \sum_{j \in i^{th} cluster} ||C_i - X_j||_2$$

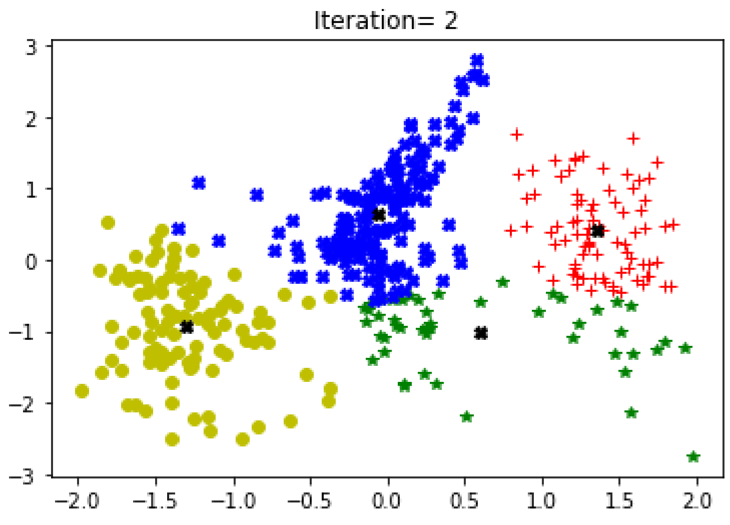
In [ ]:

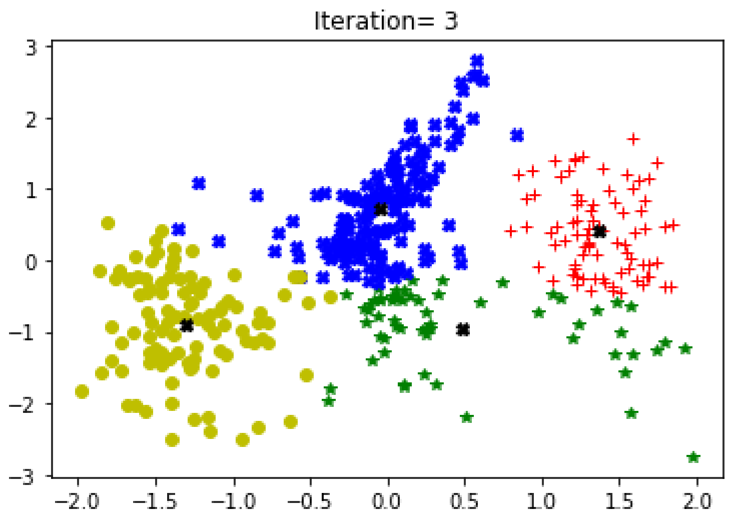
```
# write Your code here
```

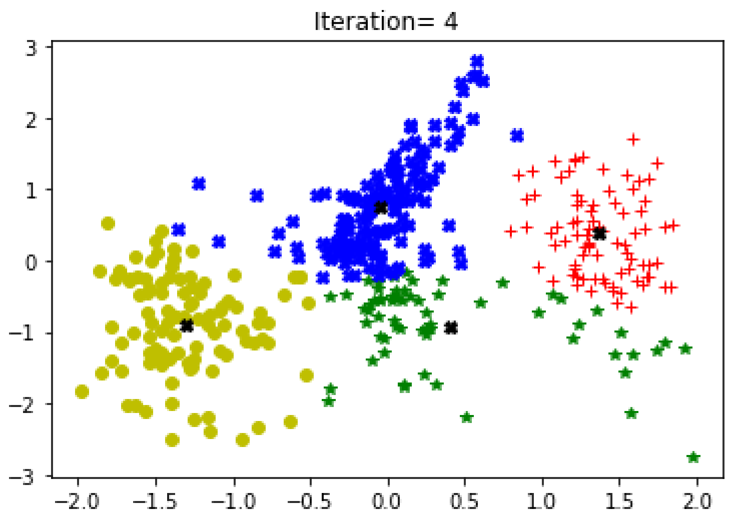
```
Out[ ]:
[<matplotlib.lines.Line2D at 0x7f2c2824a748>]
```

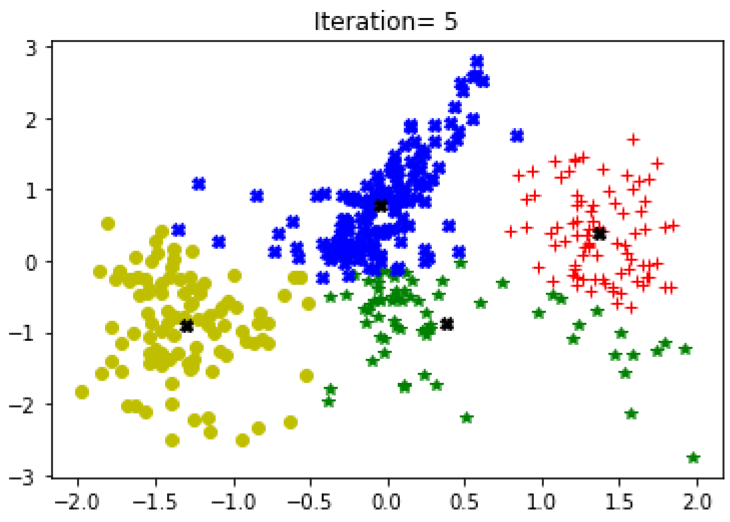


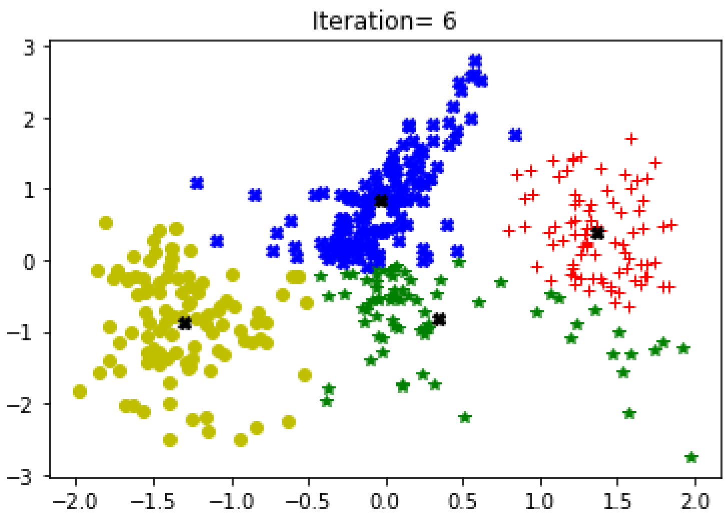


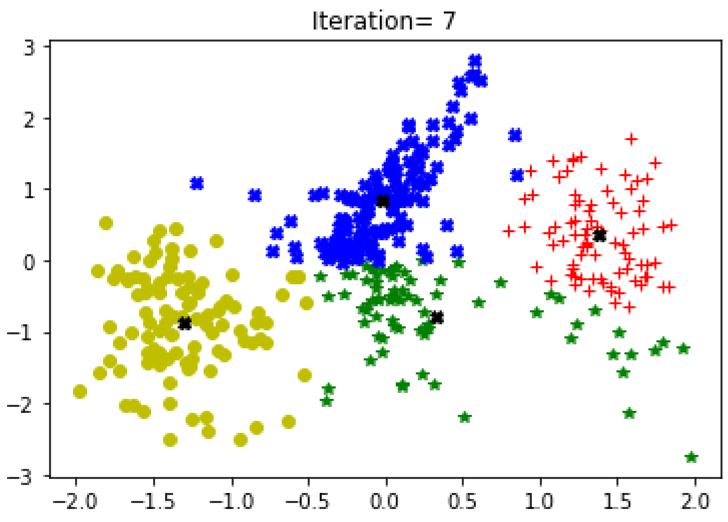


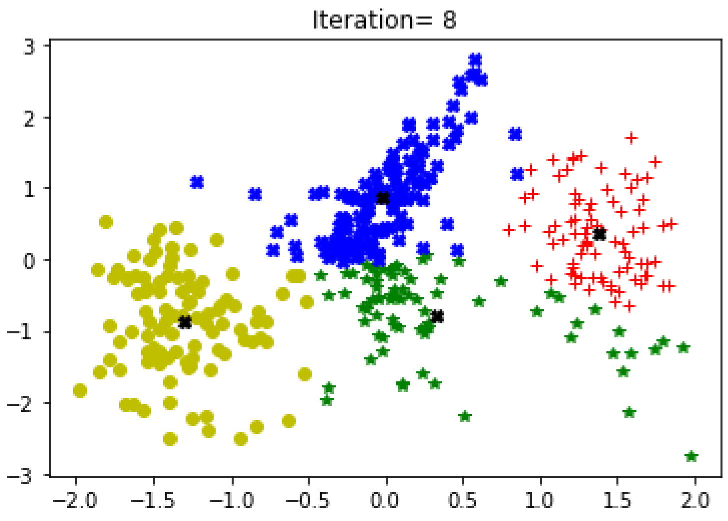

Iteration= 3
3
2
1
0
−1
−2
−3
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

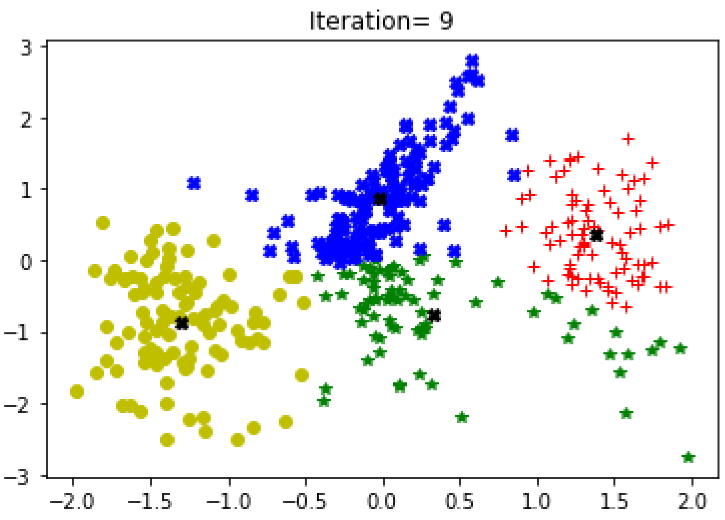

Iteration= 4
3
2
1
0
−1
−2
−3
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

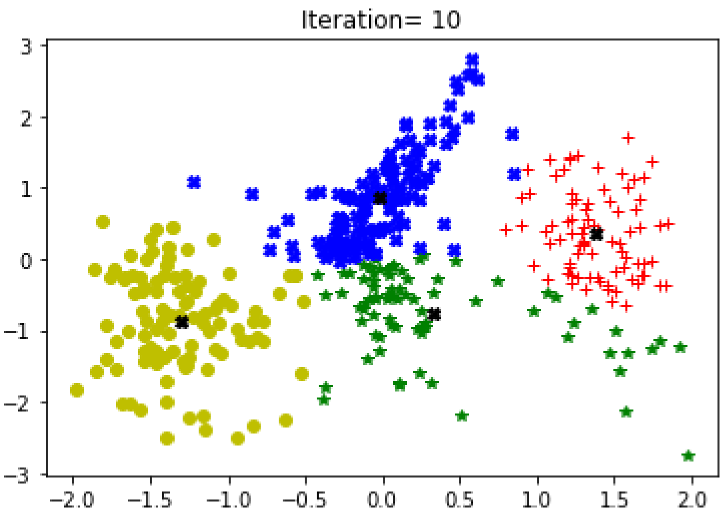

Iteration= 5
3
2
1
0
−1
−2
−3
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

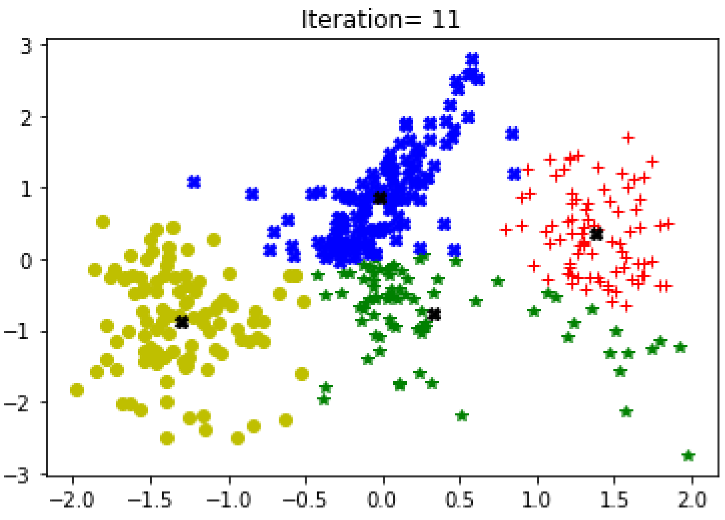

Iteration= 6
3
2
1
0
−1
−2
−3
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

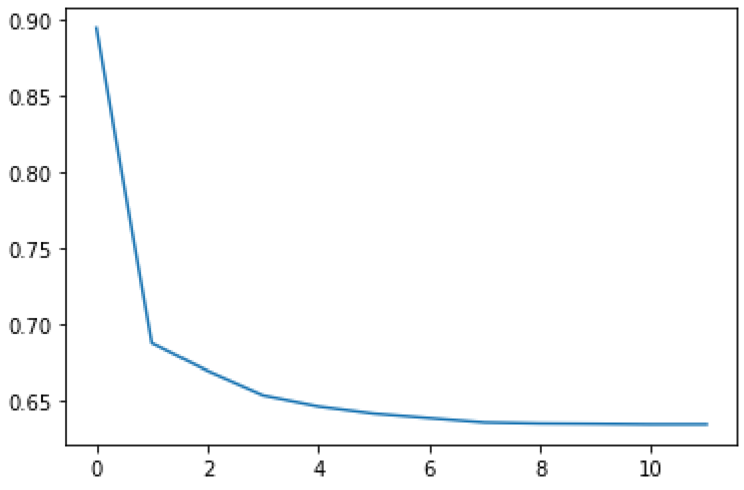

Iteration= 7
3
2
1
0
−1
−2
−3
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Iteration= 8
3
2
1
0
−1
−2
−3
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Iteration= 9
3
2
1
0
−1
−2
−3
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Iteration= 10
3
2
1
0
−1
−2
−3
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Iteration= 11
3
2
1
0
−1
−2
−3
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0



2. GMM Clustering

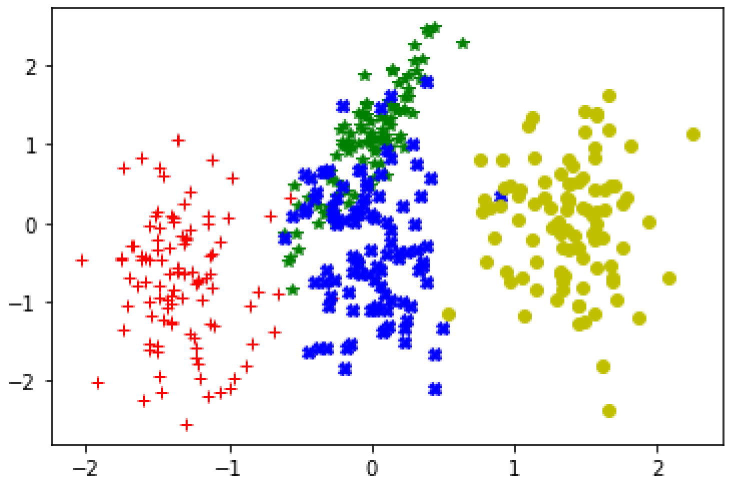
1. Data generation

a) Use the same data that you generated for K-means

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt

## Data generation
# write your code here
```

```
(400,)
```



2. Initialization

a) Mean vector (randomly any from the given data points) (μ_k)

b) Coveriance (initialize with (identity matrix)*max(data)) (Σ_k)

c) Weights (uniformly) (w_k), with constraint: $\sum_{k=1}^{K} w_k = 1$

In [ ]:

```
#%% Initialisations

def initialization(data,K):

  # write your code here
  return theta
```

3. Expectation stage

$$\gamma_{ik} = \frac{w_k P(x_i|\Phi_k)}{\sum_{k=1}^{K} w_k P(x_i|\Phi_k)}$$

where,

$$\Phi_k = \{\mu_k, \Sigma_k\}$$
$$\theta_k = \{\Phi_k, w_k\}$$
$$w_k = \frac{N_k}{N}$$
$$N_k = \sum_{i=1}^{N} \gamma_{ik}$$
$$P(x_i|\Phi_k) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} e^{-(x_i-\mu_k)^T \Sigma_k^{-1}(x_i-\mu_k)}$$

In [ ]:

```
# Expectation stage

#%% E-Step GMM
from scipy.stats import multivariate_normal
def E_Step_GMM(data,K,theta):

    # write your code here

    return responsibility
```

3. Maximization stage

a) $w_k = \frac{N_k}{N}$, where $N_k = \sum_{i=1}^{N} \gamma_{ik}$

b) $\mu_k = \frac{\sum_{i=1}^{N} \gamma_{ik} x_i}{N_k}$

c) $\Sigma_k = \frac{\sum_{i=1}^{N} \gamma_{ik} (x_i - \mu_k)(x_i - \mu_k)^T}{N_k}$

Objective function(maximized through iteration):

$$L(\theta) = \sum_{i=1}^{N} log \sum_{k=1}^{K} w_k P(x_i|\Phi_k)$$

In [ ]:

```
# Maximization stage

#%% M-STEP GMM
def M_Step_GMM(data,responsibility):
    # write your code here

    return theta, log_likelihood
```

4. Final run (EM algorithem)

a) initialization

b)Itterate E-M untill $L(\theta_n) - L(\theta_{n-1}) \leq th$

c) Plot and see the cluster allocation at each itteration

In [ ]:

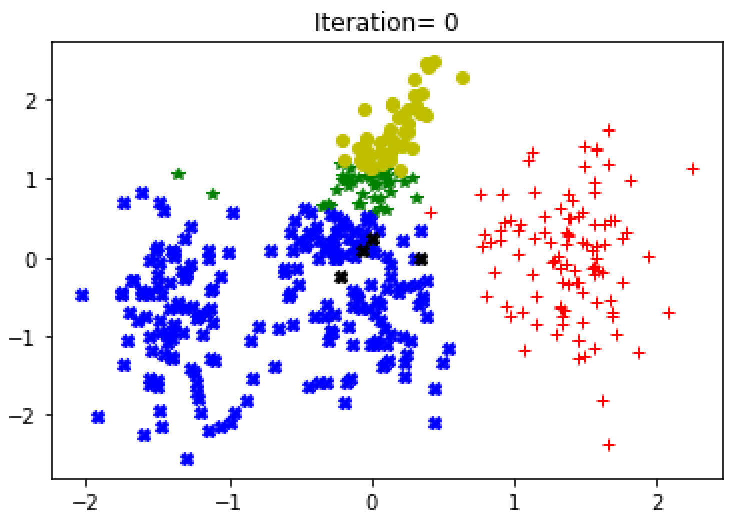
```
log_l=[]
Itr=50
eps=10**(-14)  # for threshold
clr=['r','g','b','y','k','m','c']
mrk=['+','*','X','o','.','<','p']


K=4   # no. of clusters

theta=initialization(data,K)

for n in range(Itr):

  responsibility=E_Step_GMM(data,K,theta)

  cluster_label=np.argmax(responsibility,axis=1) #Label Points

  theta,log_likhd=M_Step_GMM(data,responsibility)

  log_l.append(log_likhd)

  plt.figure()
  for l in range(K):
    id=np.where(cluster_label==l)
    plt.plot(data[id,0],data[id,1],'.',color=clr[l],marker=mrk[l])
  Cents=theta[0].T
  plt.plot(Cents[:,0],Cents[:,1],'X',color='k')
  plt.title('Iteration= %d' % (n))

  if n>2:
    if abs(log_l[n]-log_l[n-1])<eps:
      break


plt.figure()
plt.plot(log_l)
```

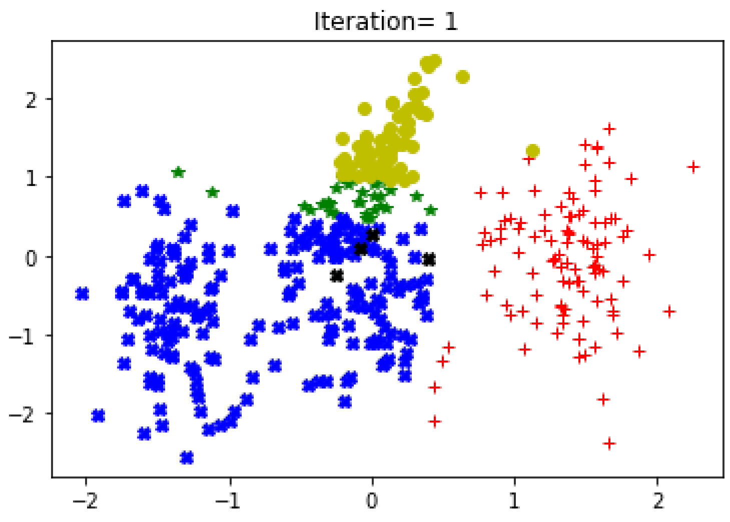
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
```

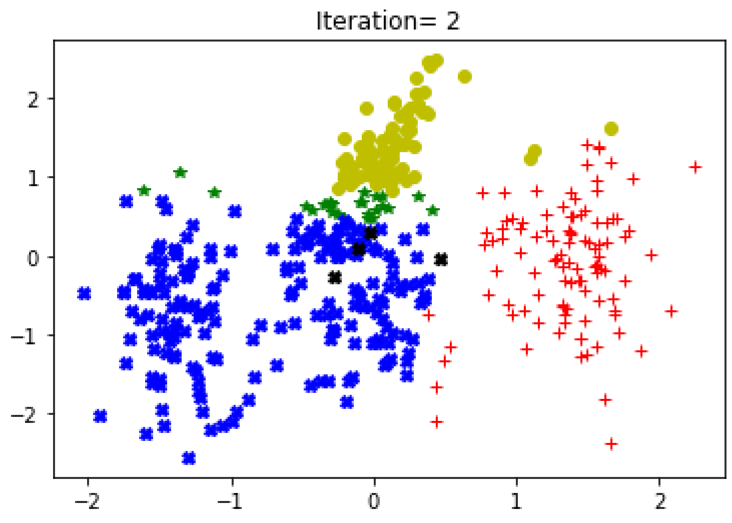
```
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
```

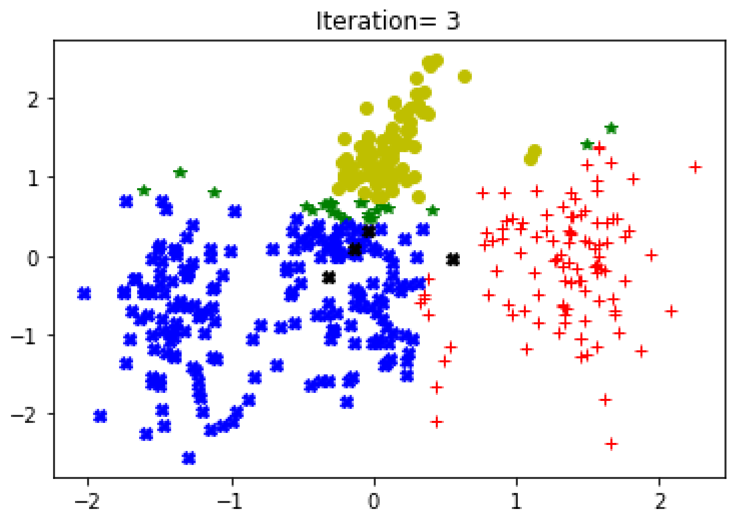
```
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:24: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:37: Run
timeWarning: More than 20 figures have been opened. Figures created
through the pyplot interface (`matplotlib.pyplot.figure`) are retain
ed until explicitly closed and may consume too much memory. (To cont
rol this warning, see the rcParam `figure.max_open_warning`).
```

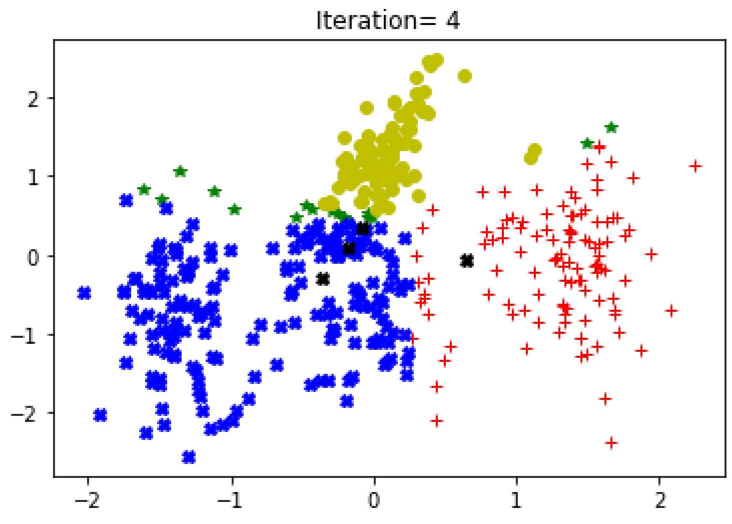
```
Out[ ]:
[<matplotlib.lines.Line2D at 0x7f2c2c5df8d0>]
```

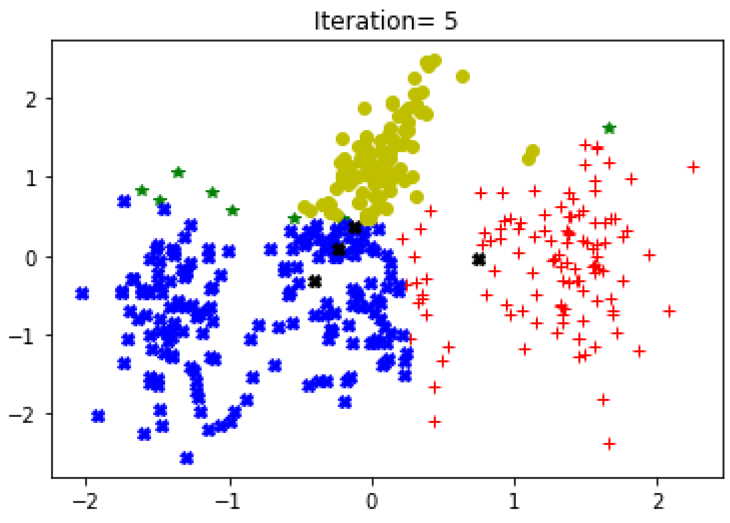


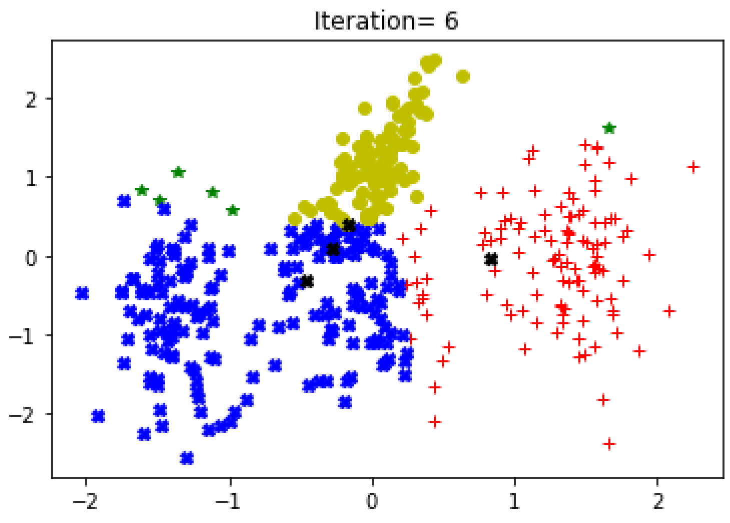


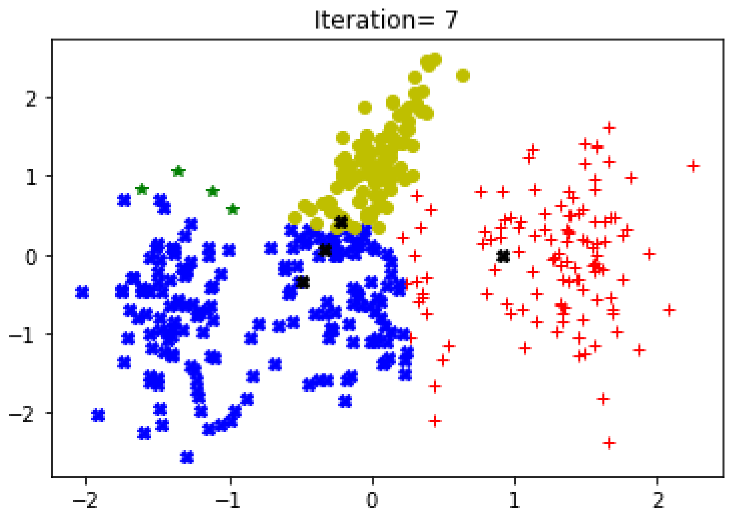

Iteration= 2
2
1
0
−1
−2
−2
−1
0
1
2


Iteration= 3
2
1
0
−1
−2
−2
−1
0
1
2

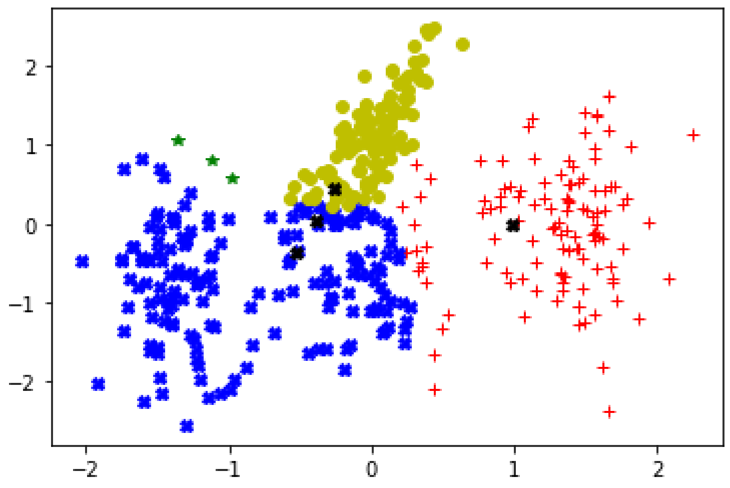

Iteration= 4
2
1
0
−1
−2
−2
−1
0
1
2


Iteration= 5
2
1
0
−1
−2
−2
−1
0
1
2

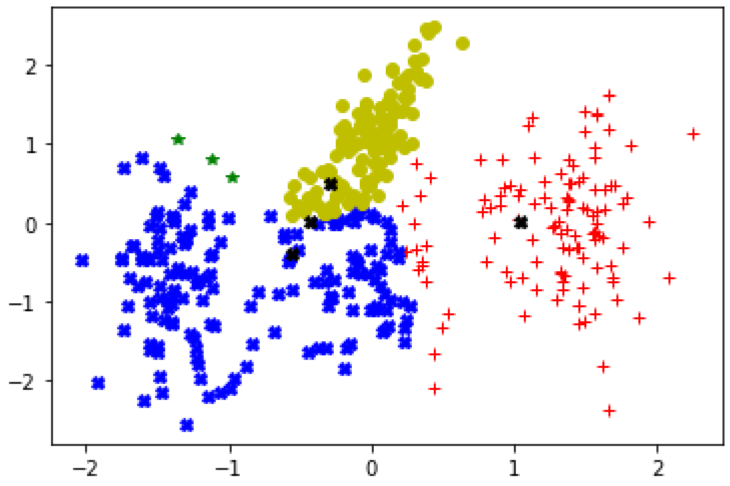

Iteration= 6
2
1
0
−1
−2
−2
−1
0
1
2


Iteration= 7
2
1
0
−1
−2
−2
−1
0
1
2

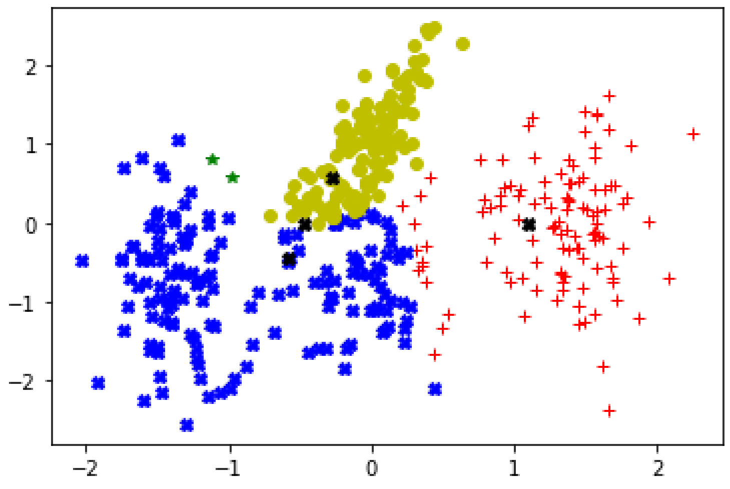
Iteration= 8

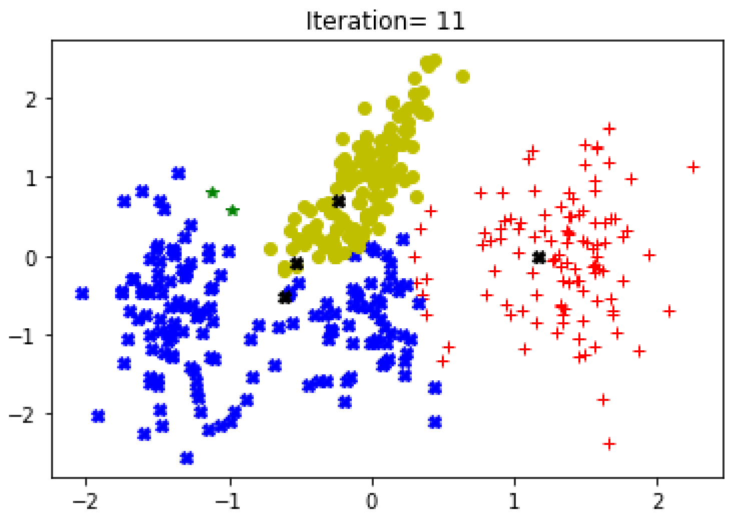


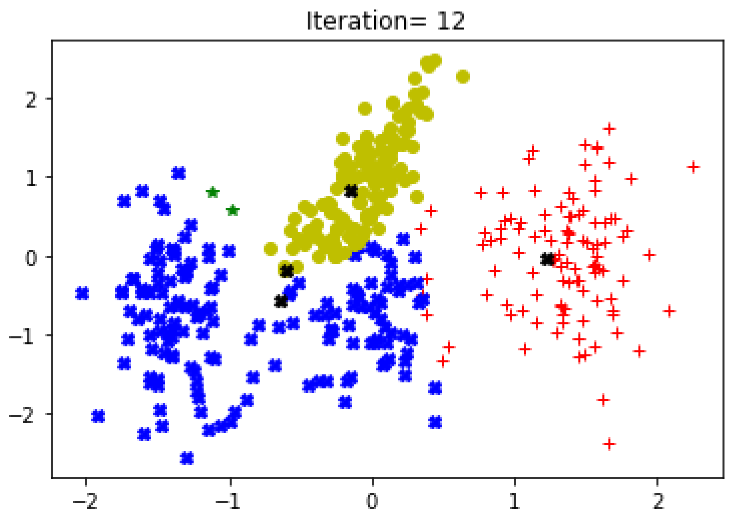
Iteration= 9

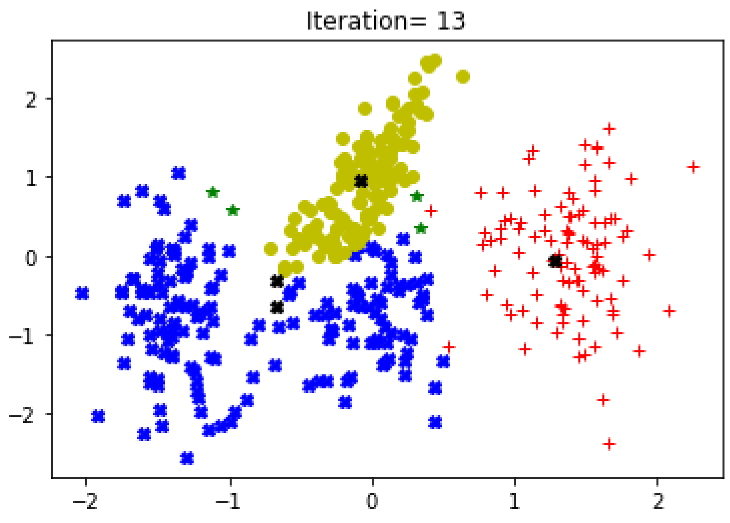


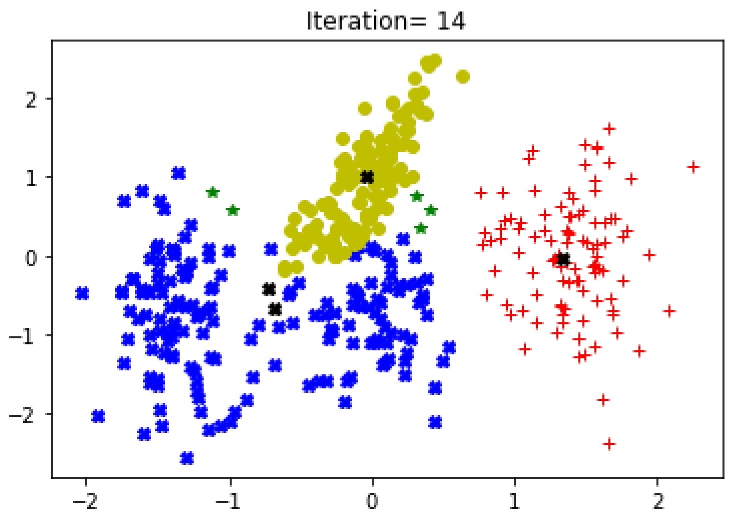
Iteration= 10

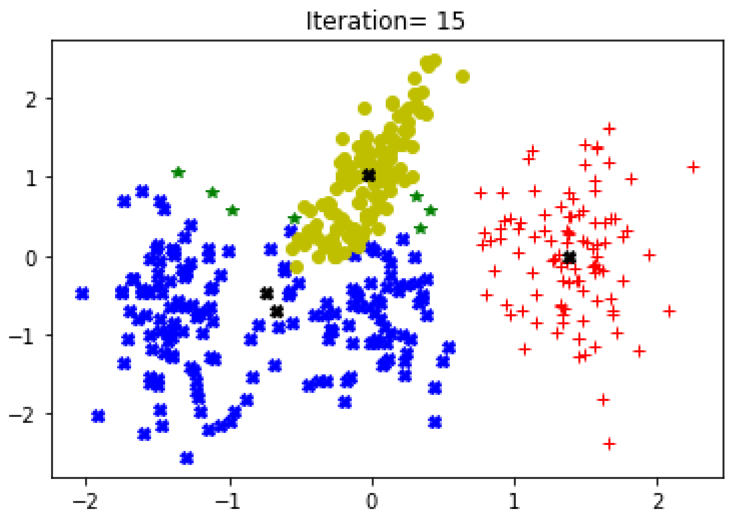


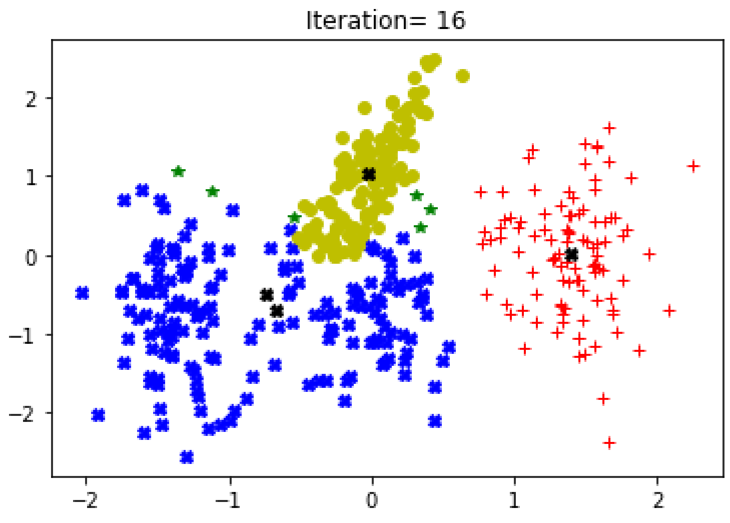

Iteration= 11
2
1
0
−1
−2
−2
−1
0
1
2

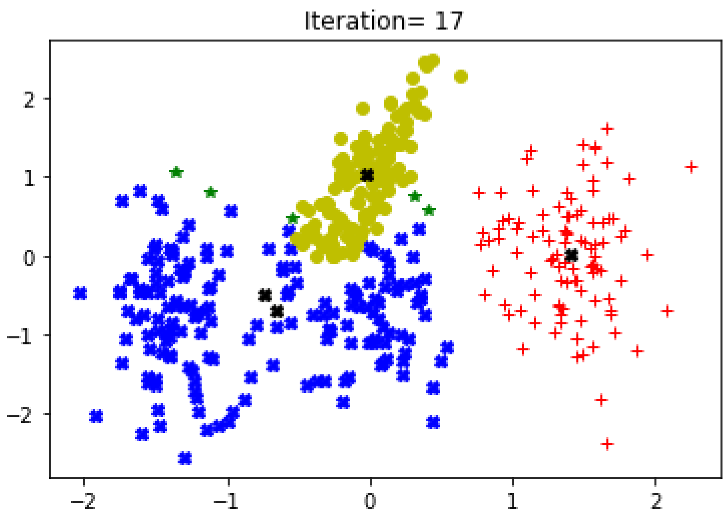

Iteration= 12
2
1
0
−1
−2
−2
−1
0
1
2

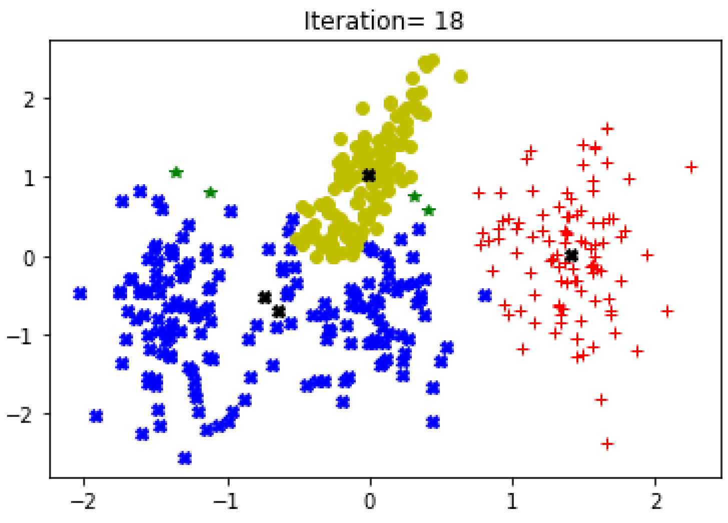

Iteration= 13
2
1
0
−1
−2
−2
−1
0
1
2

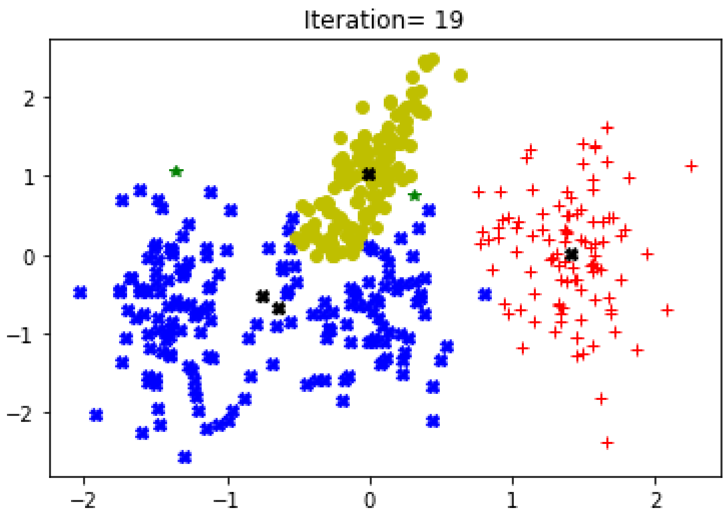

Iteration= 14

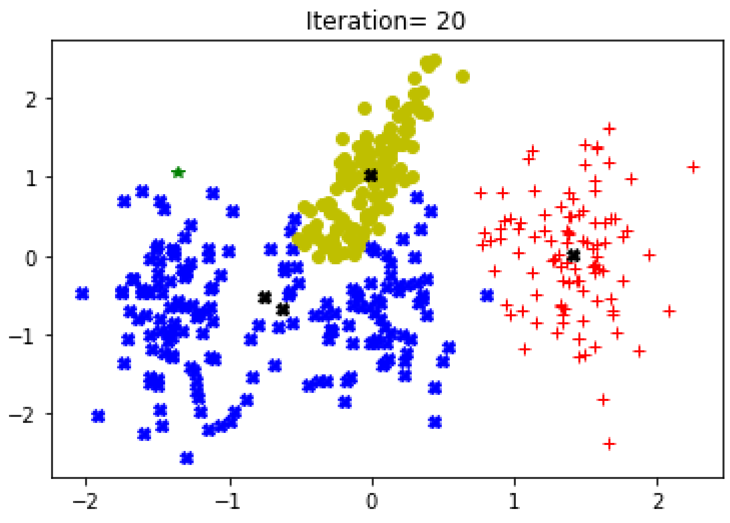

Iteration= 15

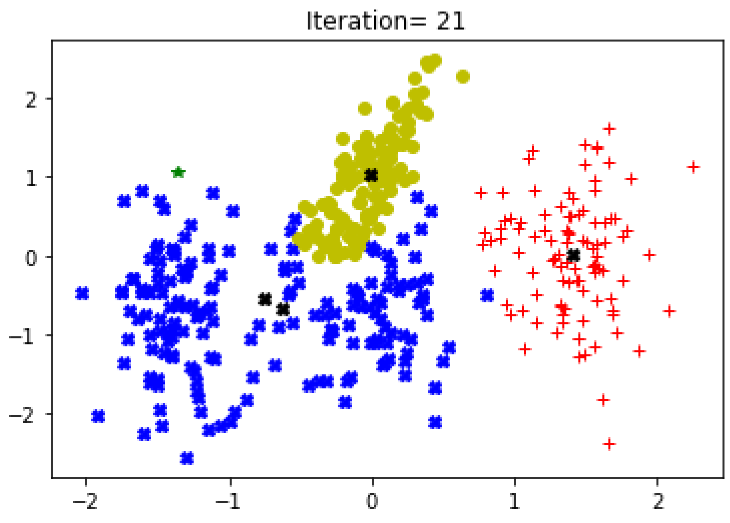

Iteration= 16

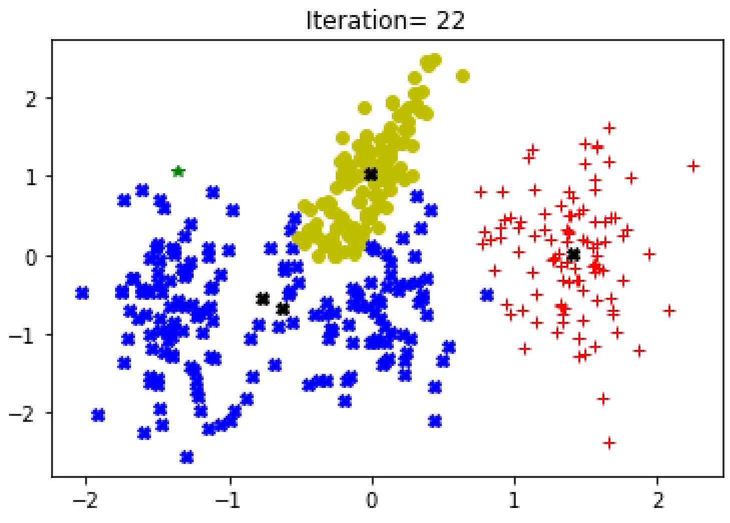

Iteration= 17
2
1
0
−1
−2
−2
−1
0
1
2

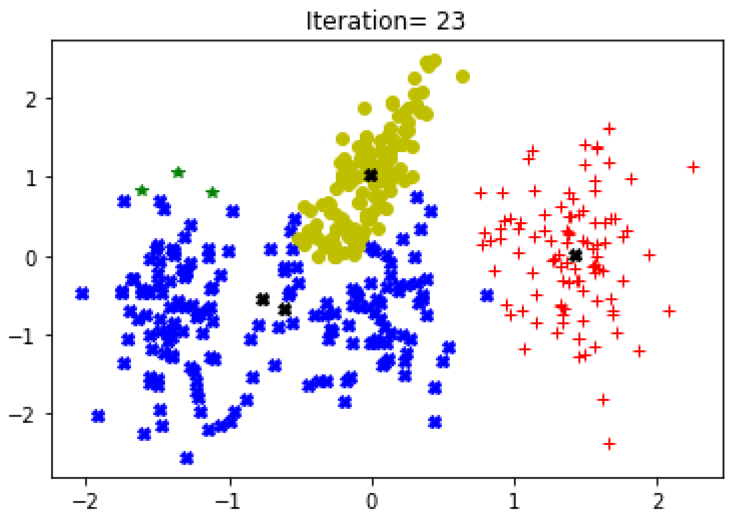

Iteration= 18
2
1
0
−1
−2
−2
−1
0
1
2

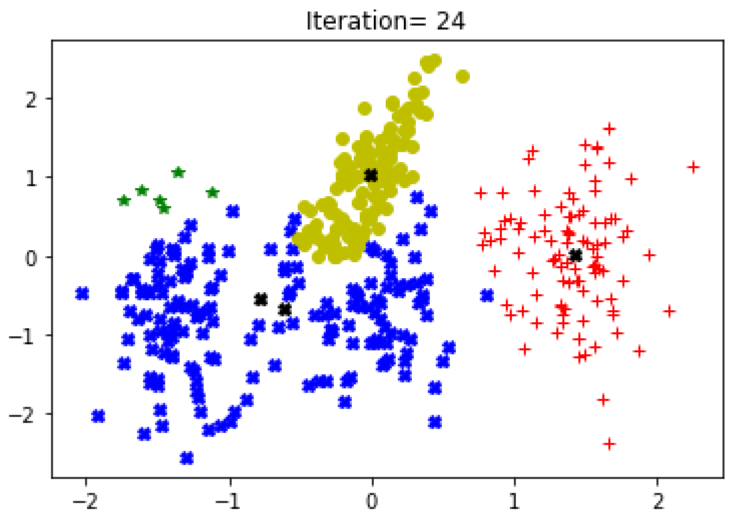

Iteration= 19
2
1
0
−1
−2
−2
−1
0
1
2

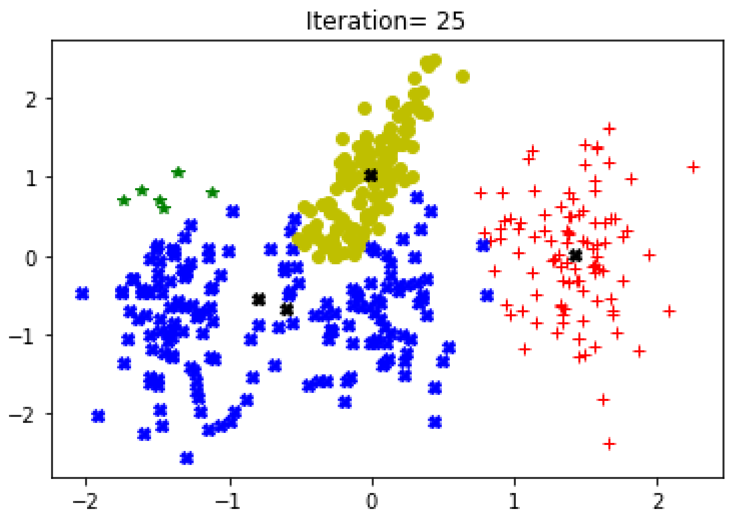

Iteration= 20
2
1
0
−1
−2
−2
−1
0
1
2

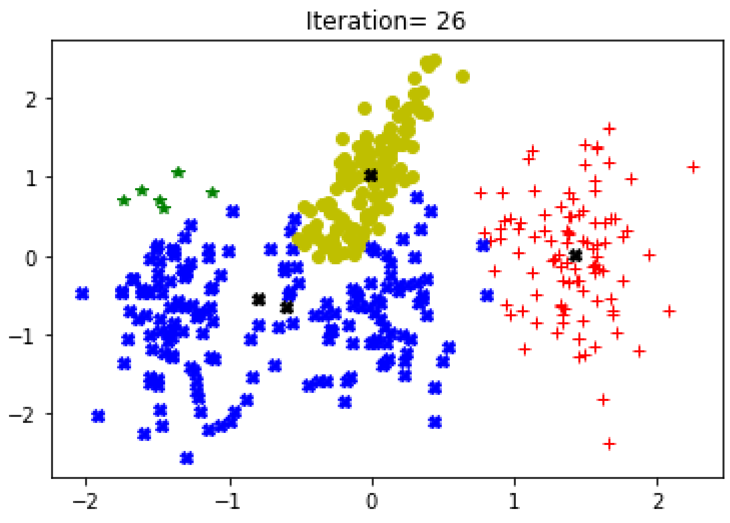

Iteration= 21
2
1
0
−1
−2
−2
−1
0
1
2

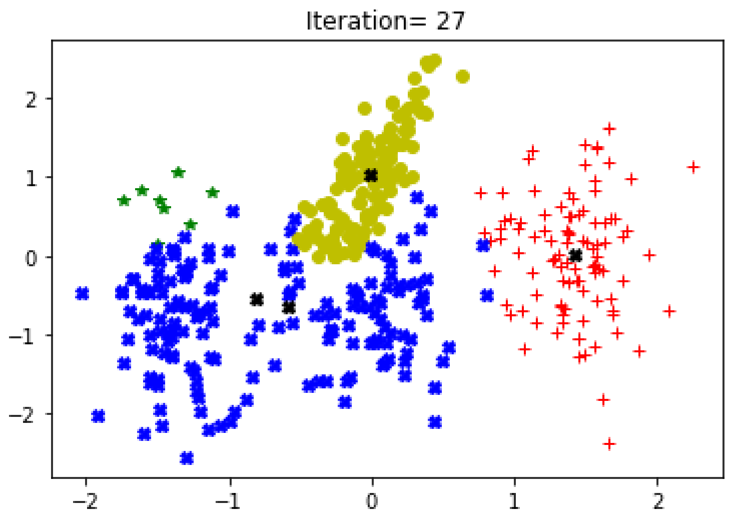

Iteration= 22
2
1
0
−1
−2
−2
−1
0
1
2

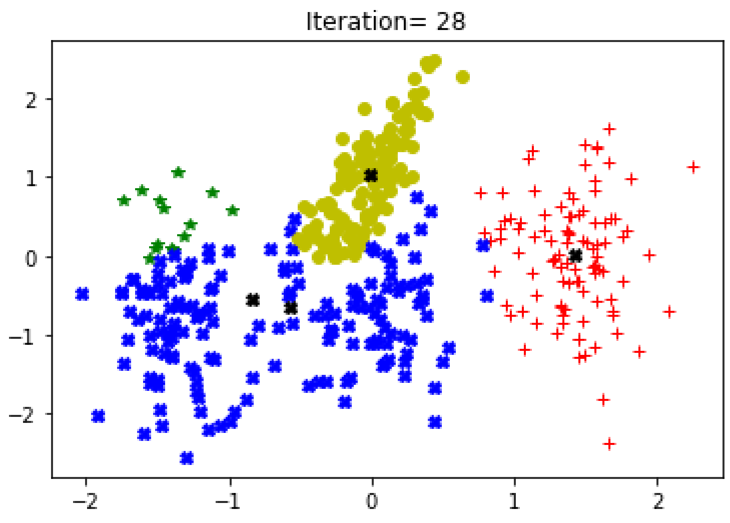

Iteration= 23
2
1
0
−1
−2
−2
−1
0
1
2

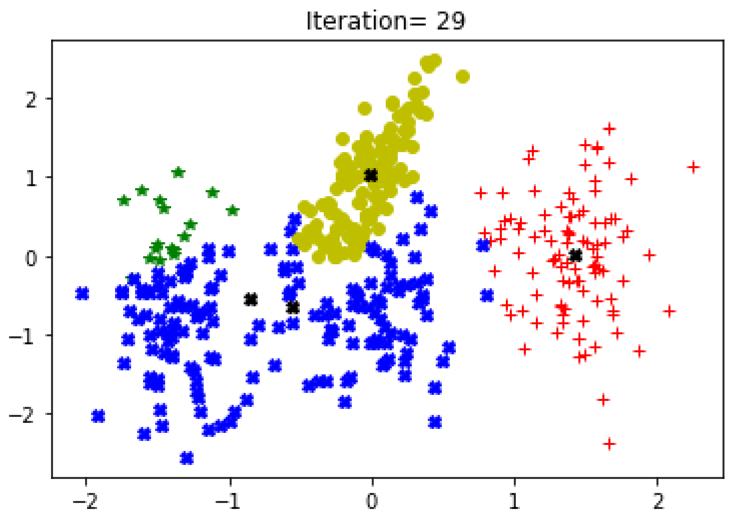

Iteration= 24
2
1
0
−1
−2
−2
−1
0
1
2

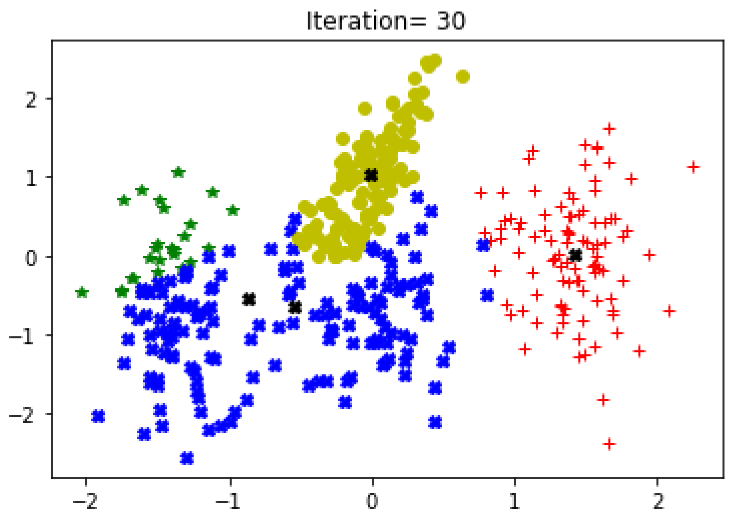

Iteration= 25
2
1
0
−1
−2
−2
−1
0
1
2

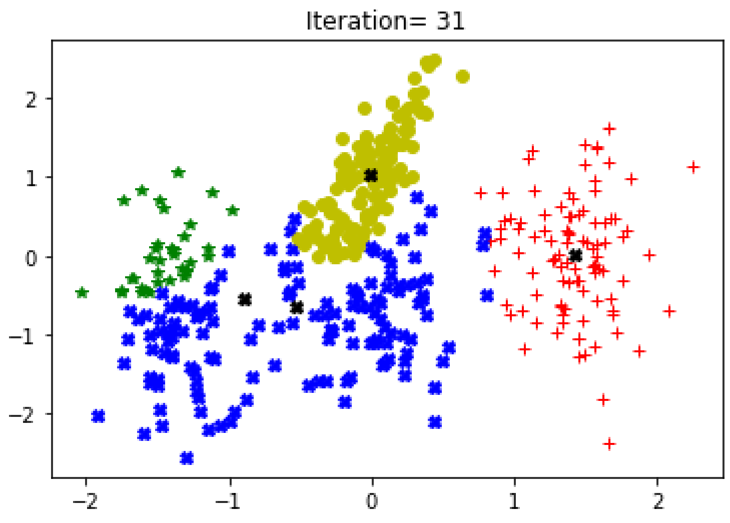

Iteration= 26
2
1
0
−1
−2
−2
−1
0
1
2


Iteration= 27
2
1
0
−1
−2
−2
−1
0
1
2

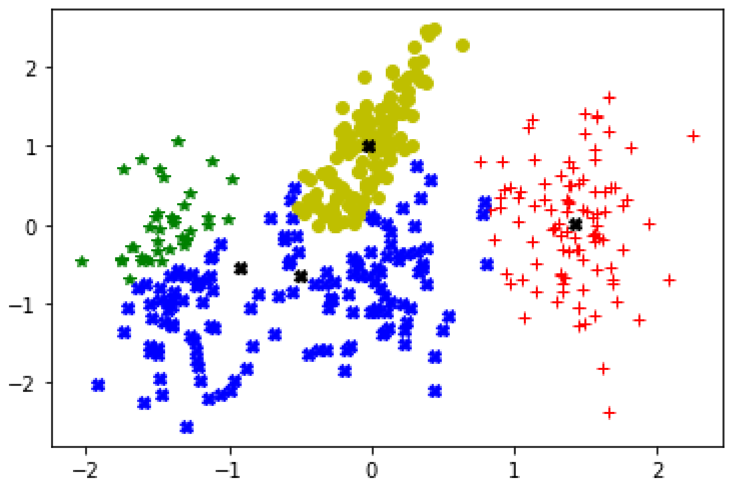

Iteration= 28
2
1
0
−1
−2
−2
−1
0
1
2


Iteration= 29
2
1
0
−1
−2
−2
−1
0
1
2

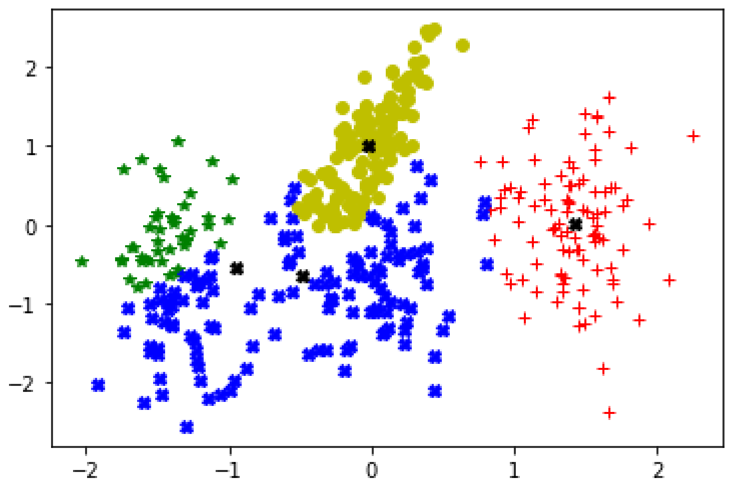

Iteration= 30
2
1
0
−1
−2
−2
−1
0
1
2


Iteration= 31
2
1
0
−1
−2
−2
−1
0
1
2

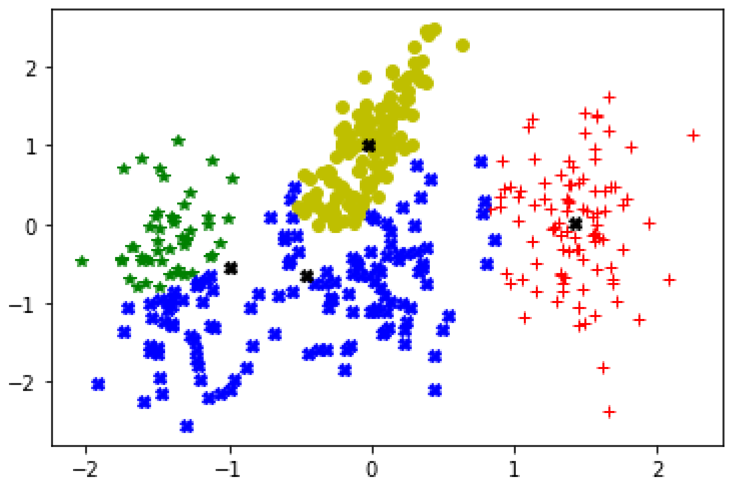
Iteration= 32

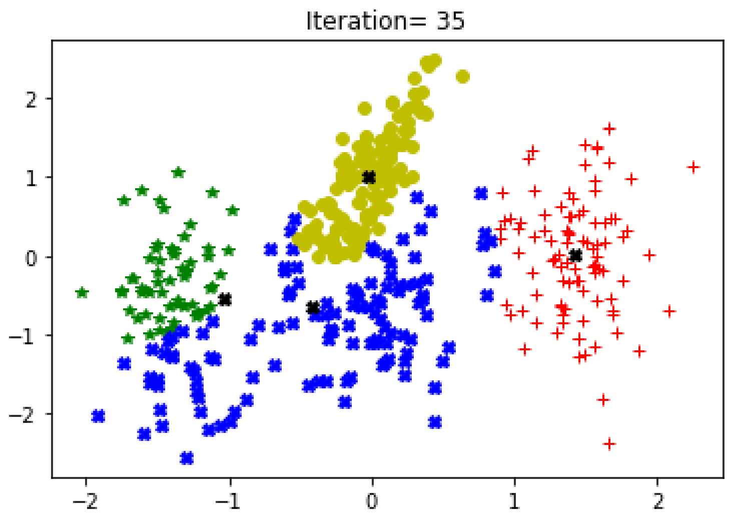


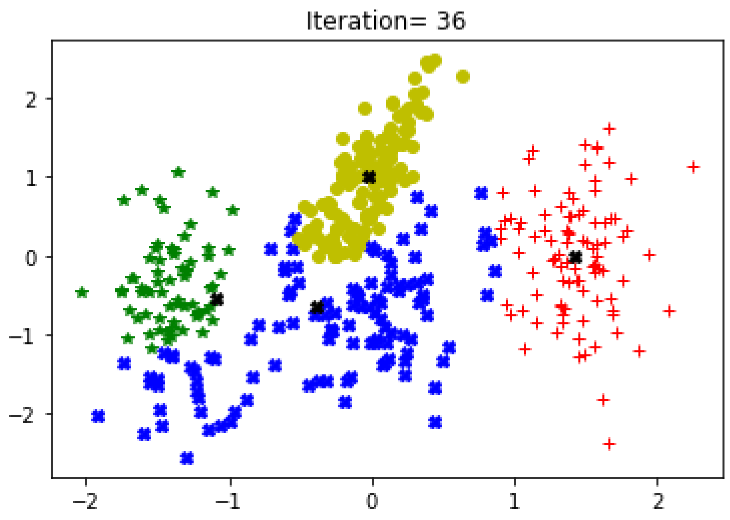
Iteration= 33

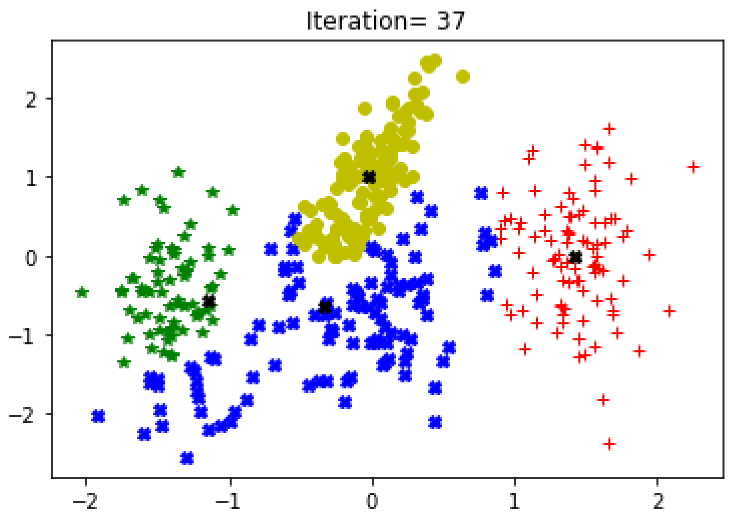


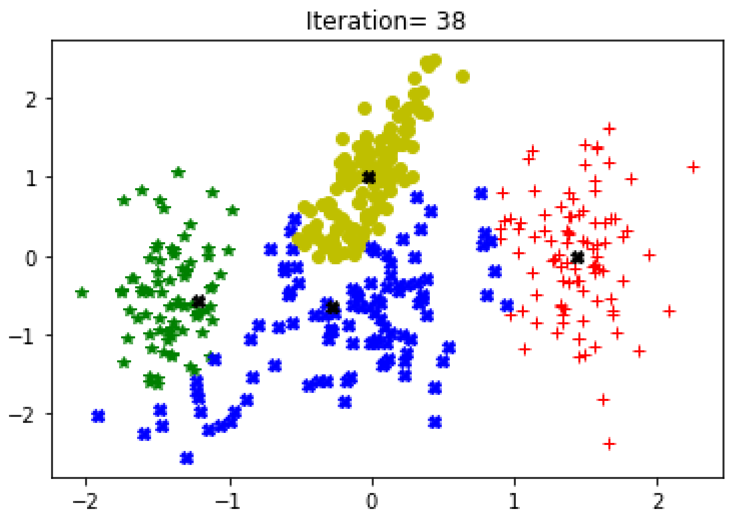
Iteration= 34

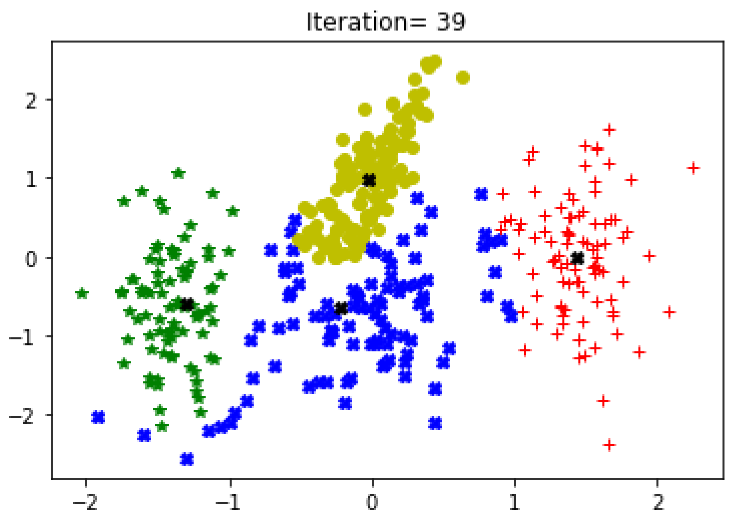


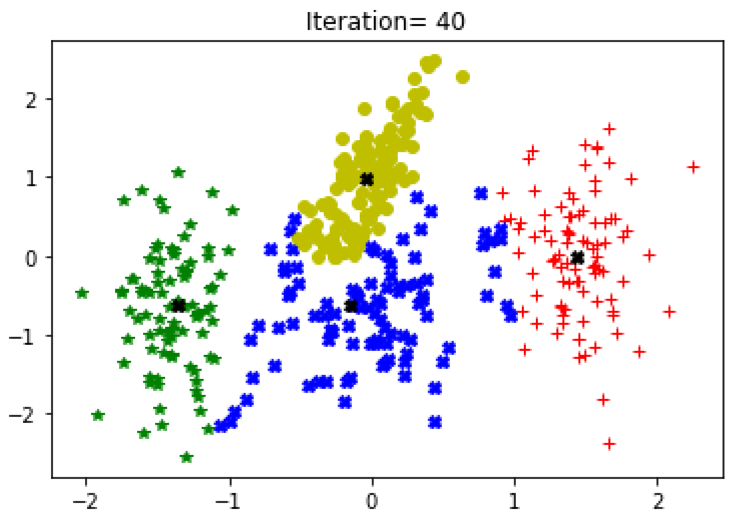

Iteration= 35
2
1
0
−1
−2
−2
−1
0
1
2

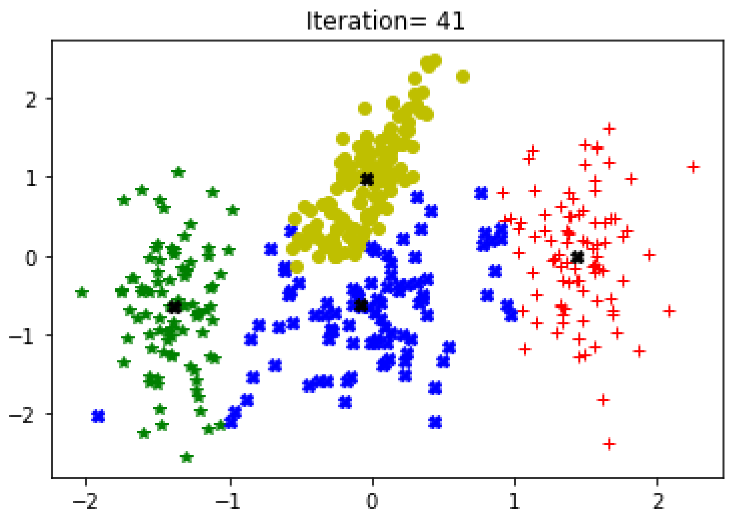

Iteration= 36
2
1
0
−1
−2
−2
−1
0
1
2

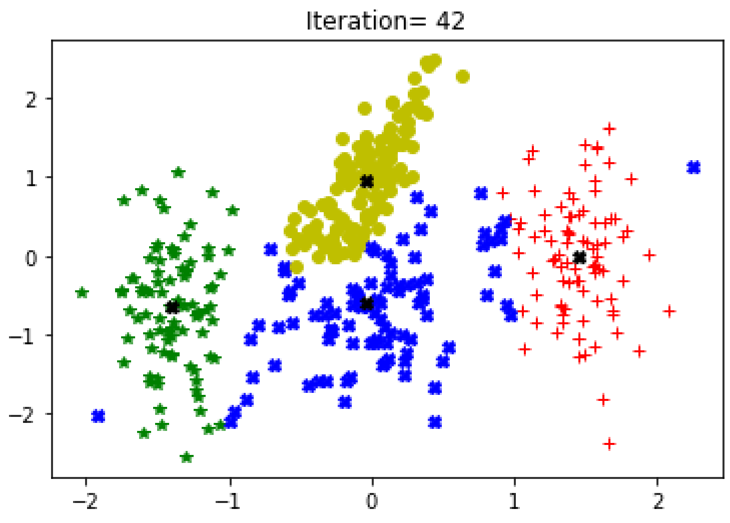

Iteration= 37
2
1
0
−1
−2
−2
−1
0
1
2

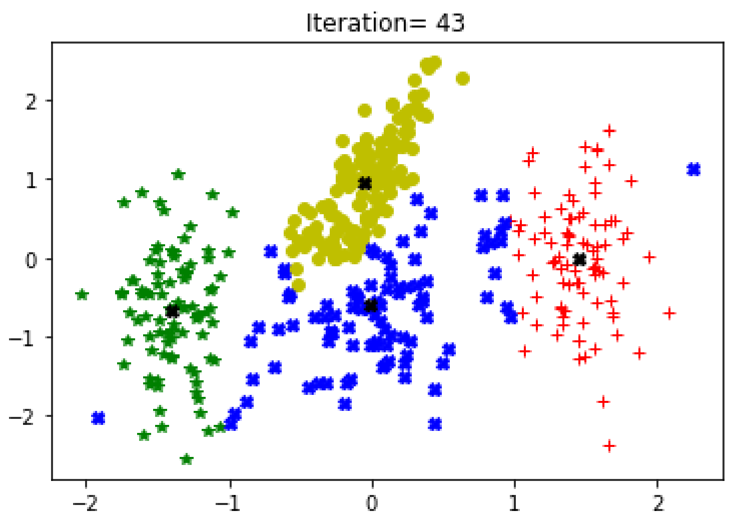

Iteration= 38
2
1
0
−1
−2
−2
−1
0
1
2

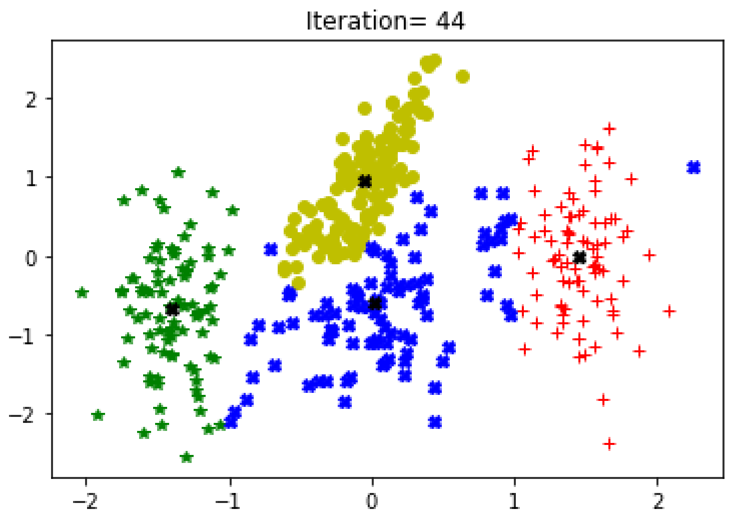

Iteration= 39
2
1
0
−1
−2
−2
−1
0
1
2

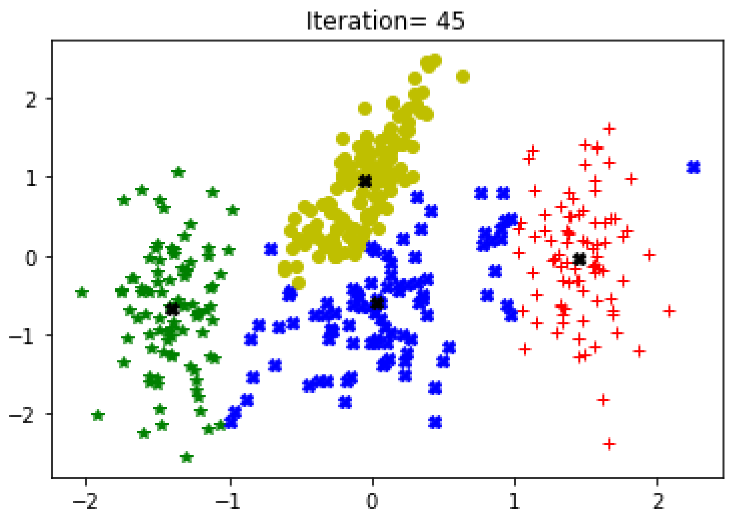

Iteration= 40
2
1
0
−1
−2
−2
−1
0
1
2

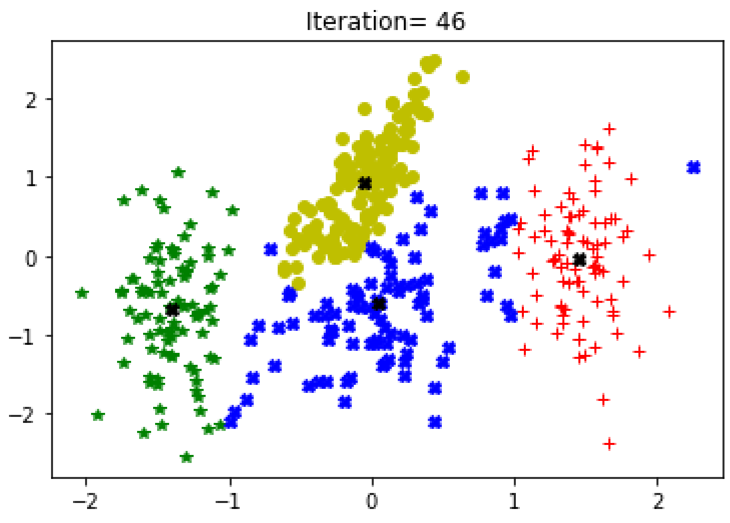

Iteration= 41
2
1
0
−1
−2
−2
−1
0
1
2

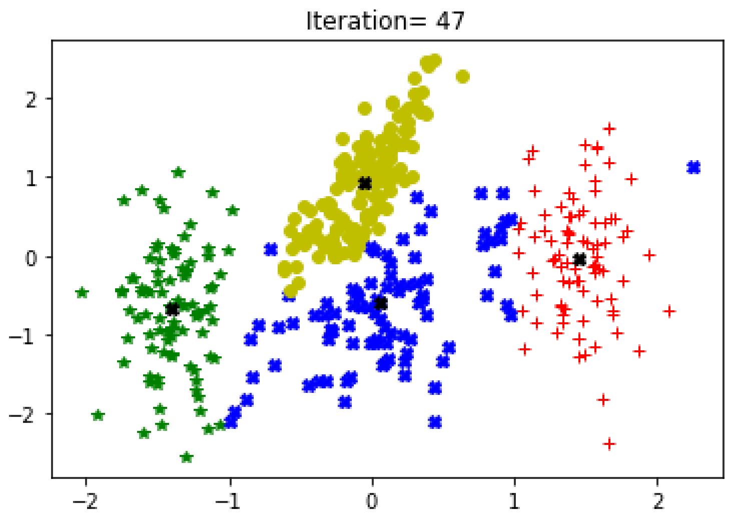

Iteration= 42
2
1
0
−1
−2
−2
−1
0
1
2

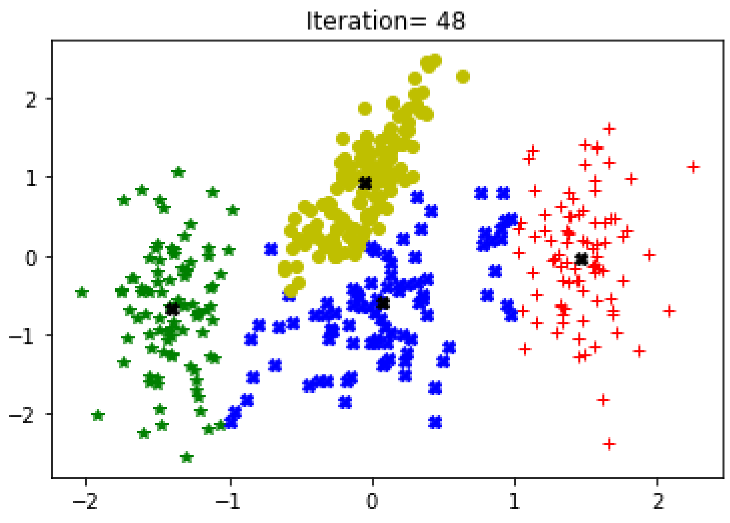

Iteration= 43
2
1
0
−1
−2
−2
−1
0
1
2

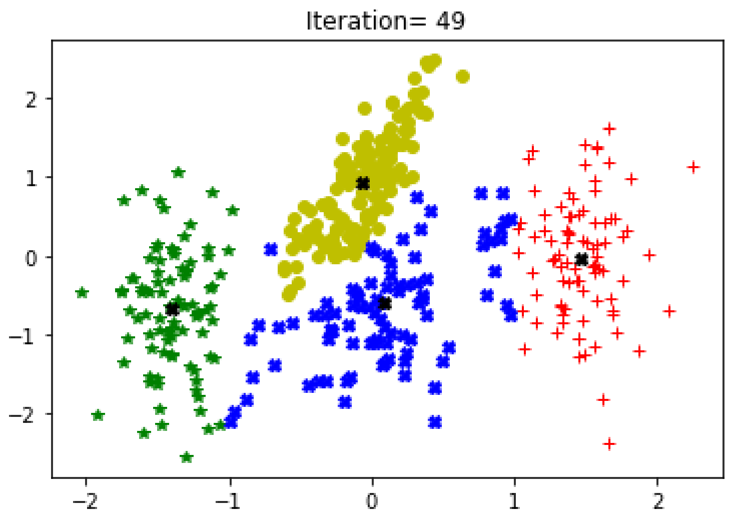

Iteration= 44
2
1
0
−1
−2
−2
−1
0
1
2

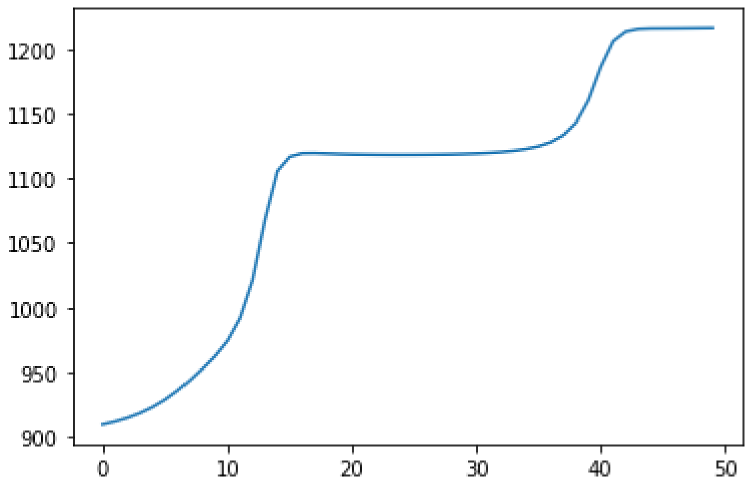

Iteration= 45
2
1
0
−1
−2
−2
−1
0
1
2


Iteration= 46
2
1
0
−1
−2
−2
−1
0
1
2


Iteration= 47
2
1
0
−1
−2
−2
−1
0
1
2


Iteration= 48
2
1
0
−1
−2
−2
−1
0
1
2


Iteration= 49
2
1
0
−1
−2
−2
−1
0
1
2



3. Write a code and report similar demonstration for Fuzzy c-means

(Note : Generate the data such that you can demonstare the drawback of K-means, and able to solve through GMM and fuzzy C-means, have to demonstrate clearly during viva)

4. Practical Example

Using K-means

a) Data preparation

1. Load Mnist data
2. Take only two class '1' and '5'

In [ ]:

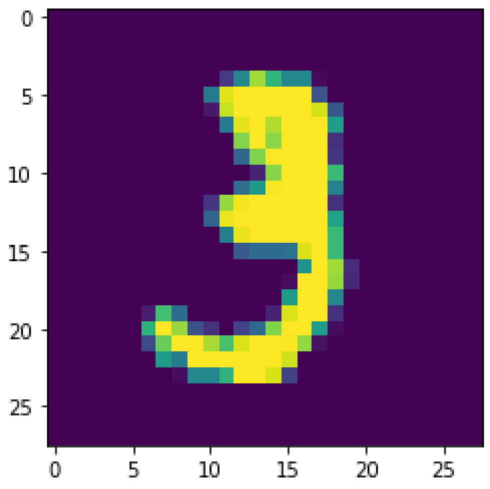
```
from google.colab import drive
drive.mount('/gdrive')
!pip install idx2numpy
```

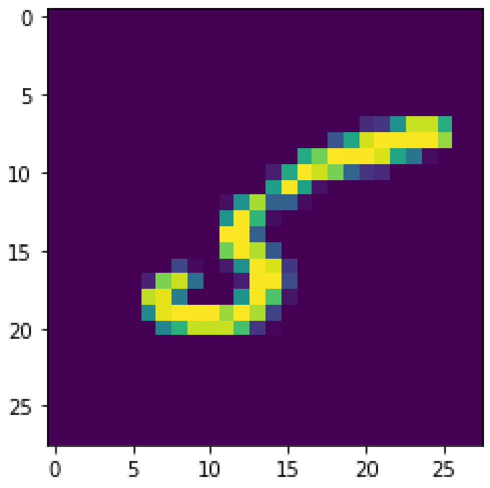
```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/au
th?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.goog
leusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&r
esponse_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fa
uth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20
https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20ht
tps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /gdrive
Collecting idx2numpy
  Downloading https://files.pythonhosted.org/packages/23/6b/abab4652
eb249f432c62431907c8de32bdcedb5abdf869ff86653efff981/idx2numpy-1.2.
2.tar.gz
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dis
t-packages (from idx2numpy) (1.18.5)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-
packages (from idx2numpy) (1.12.0)
Building wheels for collected packages: idx2numpy
  Building wheel for idx2numpy (setup.py) ... done
  Created wheel for idx2numpy: filename=idx2numpy-1.2.2-cp36-none-an
y.whl size=8032 sha256=4bbedaa71de8e46142651bc299b9bd3d205fa5ffd6285
b6dcbc9e0b516f64d0e
  Stored in directory: /root/.cache/pip/wheels/7a/b5/69/3e0757b30866
07e95db70661798fdf98a77a0bb79c54e1f320
Successfully built idx2numpy
Installing collected packages: idx2numpy
Successfully installed idx2numpy-1.2.2
```

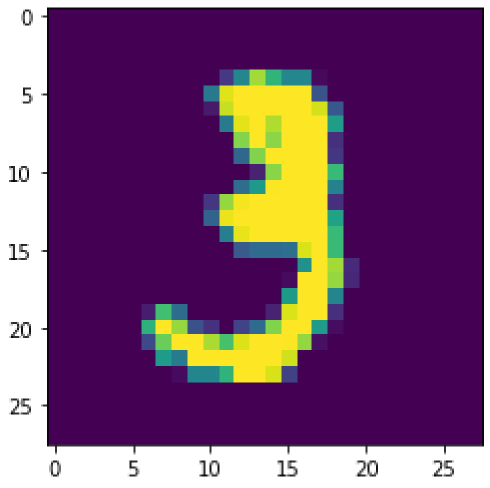
In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt

# write you code here
```

```
(11552, 784)
(11552,)
(11552, 784)
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```







1. Write a function of Kmeans as written earlier

In [ ]:

```
# k-means

def K_means_clustering(data,K,itr,eps):
  # random initialization of clusters
# write your code here


  for n in range(itr):
# assignment stage
   # write your code here

# re-estimation stage
   for i in range(K):
      # write your code here



   error.append(np.mean(DAL[:,K+1]))
   #print(Cents)



   if n>2:
      if abs(error[n]-error[n-1])<eps:
         break

   print(n)

  return DAL, Cents,error
```

1. Call the K-means function and plot the mean vectors of the cluster

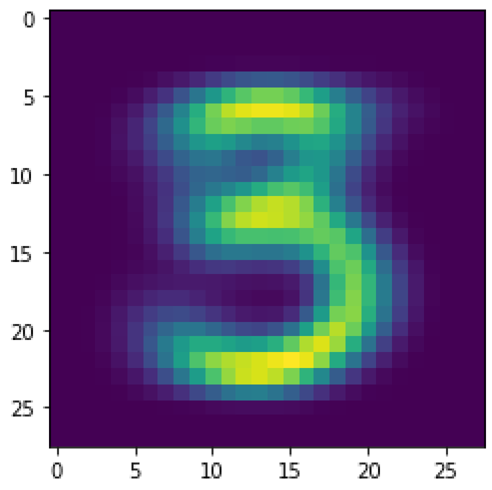
In [ ]:

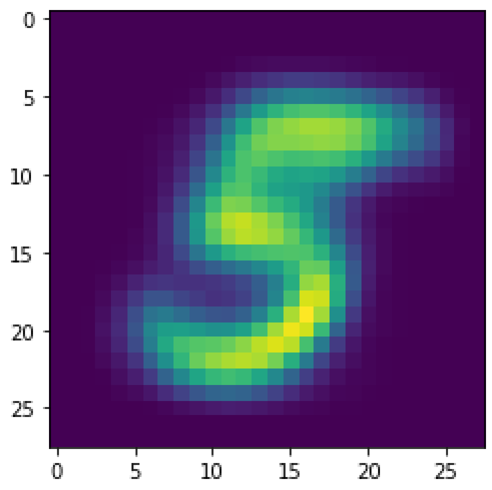
```
DAL,cents,error=K_means_clustering(data,2,200,10**(-20))

plt.imshow(np.reshape(cents[0,:],(28,28)))
plt.figure()
plt.imshow(np.reshape(cents[1,:],(28,28)))

plt.figure()
plt.plot(error)
```

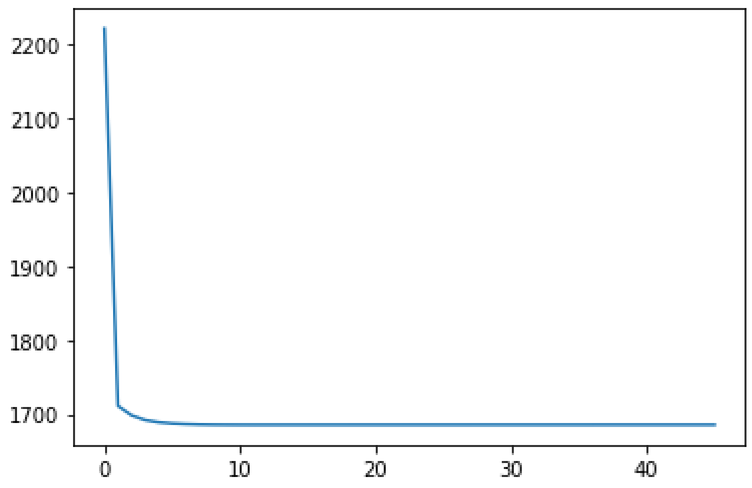
```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f2c28ffca20>]
```







5. Perform the same task for GMM and fuzzy c-means

6. Repeat the same for 3 class and perform the K-means, GMM and Fuzzy c-means clustering

7. Perform DBSCAN and show the advantages of DBSCAN over model and distance based clustering.

expected: (should visualize the cluster pattern that Model and distance based clustering can not able to capture but can be captured through DBSCAN)

8. Hierarchical Clustering

Hierarchical clustering is an unsupervised clustering technique which groups together the unlabelled data of similar characteristics.

There are two types of hierarchical clustering:

- Agglomerative Clustering
- Divisive Clustering

Agglomerative Clustering:

In this type of hierarchical clustering all data set are considered as indivisual cluster and at every iterations clusters with similar characteristics are merged to give bigger clusters. This is repeated untill one single cluster is reached. It is also called bottem-top approach.

Divisive Clustering:

It is an opposite of Agglomerative clustering. In this we start from one cluster which contains all data points in one. Iteratively we separate all the cluster of points which aren't similar in characteristics. It is also called top-bottom approach.

Agglomerative Clustering:

Lets start with some domy example :

X=$[x_1, x_2, \ldots, x_5]$, with

$$x_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, x_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, x_3 = \begin{bmatrix} 5 \\ 4 \end{bmatrix}, x_4 = \begin{bmatrix} 6 \\ 5 \end{bmatrix}, x_5 = \begin{bmatrix} 6.5 \\ 6 \end{bmatrix}$$

Steps to perform Agglomerative Clustering:

1. Compute Distance matrix ($N \times N$ matrix, where N number of vectors present in the dataset): $D(a,b) = ||x_a - x_b||_2$
2. Replace the diagonal elements with inf and find the index of the minimum element present in the distance matrix (suppose we get the location (l,k)).
3. Replace $x_{min(l,k)} = .5 \times [x_l + x_m]$ and delete $x_{max(l,m)}$ vector from X(i.e now $(N = N - 1)$),

repeat from step 1 again untill all the vectors combined to a single cluster.

In [1]:

```
import numpy as np
def Euclidian_Dist(x,y):

  return #write your code here

def Dist_mat(X):
 #write your code here
  return dist_mat

def combine(X):
  #write your code here

  return newX
```

In [3]:

```
X=np.array([[1,1],[2,1],[5,4],[6,5],[6.5,6]])
X=X.transpose()

#write your code here
## velidate from inbuilt Dendogram

import plotly.figure_factory as ff



lab=np.linspace(1,X.shape[1],X.shape[1])
fig = ff.create_dendrogram(X.T, labels=lab)
fig.update_layout(width=800, height=300)
fig.show()
```

```
[[1.  2.  5.  6.  6.5]
 [1.  1.  4.  5.  6. ]]
[[inf 1.  5.  6.4 7.4]
 [1.  inf 4.2 5.7 6.7]
 [5.  4.2 inf 1.4 2.5]
 [6.4 5.7 1.4 inf 1.1]
 [7.4 6.7 2.5 1.1 inf]]
Vector of X to be combined:  [1 2]

Mean of clusters after every iteration:

 [[1.5 5.  6.  6.5]
 [1.  4.  5.  6. ]]
[[inf 4.6 6.  7.1]
 [4.6 inf 1.4 2.5]
 [6.  1.4 inf 1.1]
 [7.1 2.5 1.1 inf]]
Vector of X to be combined:  [3 4]

Mean of clusters after every iteration:

 [[1.5  5.   6.25]
 [1.   4.   5.5 ]]
[[inf 4.6 6.5]
 [4.6 inf 2. ]
 [6.5 2.  inf]]
Vector of X to be combined:  [2 3]

Mean of clusters after every iteration:

 [[1.5   5.625]
 [1.    4.75 ]]
[[inf 5.6]
 [5.6 inf]]
Vector of X to be combined:  [1 2]

Mean of clusters after every iteration:

 [[3.5625]
 [2.875 ]]

cluster combination order:

 [array([1, 2]), array([3, 4]), array([2, 3]), array([1, 2])]
```

Divisive clustering:

It is a top down approach of hierarchial clustering

Lets start with some domy example :

X=$[x_1, x_2, \ldots, x_5]$, with

$x_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $x_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $x_3 = \begin{bmatrix} 5 \\ 4 \end{bmatrix}$, $x_4 = \begin{bmatrix} 6 \\ 5 \end{bmatrix}$, $x_5 = \begin{bmatrix} 6.5 \\ 6 \end{bmatrix}$

1. Find the biggest cluster (having highest diameter), initially the single cluster is the biggest cluster.

$$Diameter_{cluster} = \max_{i,j} ||x_i - x_j||_2$$

i, j will move over all the elements in the cluster.

1. find the splinter element of the cluster by using the maximum average distance between the other elements.

$$d_k = \frac{1}{N-1} \sum_{i=1}^{N} ||x_k - x_i||_2$$
$$splinter - group - element = arg\max_{1 \le k \le N}(d_k)$$

repeat the same and assign element to the splinter group untill the differance between average incluster distance and average splinter group distance of each element turns negative.

$$d_{avgsplint_k} = \frac{1}{M-1} \sum_{i=1}^{M} ||x_k - x_i||_2$$

Stop:

$$d_k - d_{avgsplint_k} < 0$$

and assign the splinter group as a new cluster.

1. Repeat the step 1 and 2 untill each cluster have only one element.

   4.**Plot the cluster split with respect to their diameter**

In [4]:

```
import numpy as np
def Euclidian_Dist(x,y):
  return #write your code here

def Dist_mat(X):
  #write your code here
  return dist_mat


def avg_distance(X):
  #write your code here
  return #write your code here
```

In [6]:

```
def get_diameter(X, i):
    """Returns the diameter of the ith cluster in X"""
    #write your code here
    return diameter

def get_biggest_cluster(X):
    """ Returns the cluster index having largest diameter"""
    #write your code here
    # index having max diameter
    return max_cluster_ind

def avg_spl_dists(cluster, splinter):
    """ Return the average of distances of each point belonging to cl wrt splint
er"""
    #write your code here
    return avg_dists
```

In [7]:

```
# Implement Divisive Clustering
import numpy as np
X = np.array([[1,1], [2,1], [5,4], [6,5], [6.5,6]])
X = X.transpose() # Shape after transpose: [2, 5]
num_points = X.shape[1]
print(f'X:\n {X}')

#write your code here
```

```
X:
 [[1.  2.  5.  6.  6.5]
 [1.  1.  4.  5.  6. ]]
Initial Number of clusters: 1
---------------- Iteraion - 1 -------------------
Biggest cluster ind is: 0
Biggest Cluster is:
 [[1.  2.  5.  6.  6.5]
 [1.  1.  4.  5.  6. ]]
Cluster:
 [[2.  5.  6.  6.5]
 [1.  4.  5.  6. ]]
 Shape: (2, 4)
Splinter:
 [[1.]
 [1.]]
 Shape: (2, 1)
New member added to splinter of index 0 and new member is
 [[2.]
 [1.]]
New cluster shape is (2, 3)
 [[5.  6.  6.5]
 [4.  5.  6. ]]
New splinter shape is (2, 2)
 [[1. 2.]
 [1. 1.]]
Final splinter and cluster shapes: (2, 2), (2, 3)
New num of clusters after splitting is: 2
[[5.  6.  6.5]
 [4.  5.  6. ]],
[[1. 2.]
 [1. 1.]],
---------------- Iteraion - 2 -------------------
Biggest cluster ind is: 0
Biggest Cluster is:
 [[5.  6.  6.5]
 [4.  5.  6. ]]
Cluster:
 [[6.  6.5]
 [5.  6. ]]
 Shape: (2, 2)
Splinter:
 [[5.]
 [4.]]
 Shape: (2, 1)
Final splinter and cluster shapes: (2, 1), (2, 2)
New num of clusters after splitting is: 3
[[1. 2.]
 [1. 1.]],
[[6.  6.5]
 [5.  6. ]],
[[5.]
 [4.]],
---------------- Iteraion - 3 -------------------
Biggest cluster ind is: 1
Biggest Cluster is:
 [[6.  6.5]
 [5.  6. ]]
Cluster:
 [[6.5]
 [6. ]]
 Shape: (2, 1)
Splinter:
 [[6.]
 [5.]]
 Shape: (2, 1)
Final splinter and cluster shapes: (2, 1), (2, 1)
New num of clusters after splitting is: 4
[[6.  6.5]
 [5.  6. ]],
[[5.]
 [4.]],
[[6.5]
 [6. ]],
[[6.]
 [5.]],
--------------- Iteraion - 4 ------------------
Biggest cluster ind is: 0
Biggest Cluster is:
 [[6.  6.5]
 [5.  6. ]]
Cluster:
 [[6.5]
 [6. ]]
 Shape: (2, 1)
Splinter:
 [[6.]
 [5.]]
 Shape: (2, 1)
Final splinter and cluster shapes: (2, 1), (2, 1)
New num of clusters after splitting is: 5
[[5.]
 [4.]],
[[6.5]
 [6. ]],
[[6.]
 [5.]],
[[6.5]
 [6. ]],
[[6.]
 [5.]],
```

9.Take a real data example and demonstrate both Agglomerative and Divisive clustering.