EE2211 Introduction to Machine Learning

T14 & T22, Chua Dingjuan elechuad@nus.edu.sg

Materials @ tiny.cc/ee2211tut

Tutorial 11

K-MEANS

Discussion of Solutions 11

Q1-5, 6,7

Question6

Generate three clusters of data using the following codes.

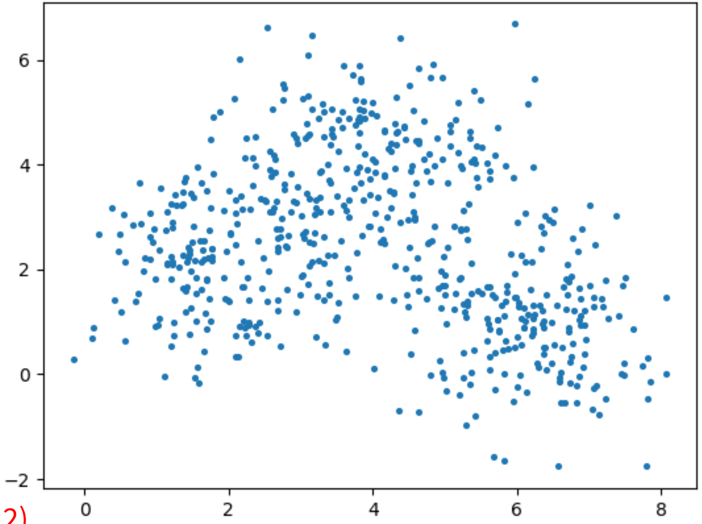
(i) Implement the Naïve K-means (the basic/standard algorithm shown in lecture) clustering algorithm to find the 3 cluster centroids. Classify the data based on the three centroids found and illustrate the results using a plot (e.g., mark the 3 clusters of data points using different colours).

(ii) Change the number of clusters K to 5 and classify the data points again with a plot illustration.

Preparation...

```
## Import necessary libraries
import random as rd
import numpy as np # linear algebra
from matplotlib import pyplot as plt
## Generate data
## Set three centers, the model should predict similar results
center_1 = np.array([2,2])
center_2 = np.array([4,4])
center_3 = np.array([6,1])
## Generate random data and center it to the three centers
data_1 = np.random.randn(200, 2) + center_1
data_2 = np.random.randn(200,2) + center_2
data_3 = np.random.randn(200,2) + center_3
data = np.concatenate((data_1, data_2, data_3), axis = 0)
plt.scatter(data[:,0], data[:,1], s=7)
```

```
X = data
m= X.shape[0] #number of training examples
d = X.shape[1] #number of features. Here d=2
```



np.random.randn(200,2)

```
array([[ 3.61944171e-01, -5.33281503e-01],
       [ 2.11683432e+00,  7.22587205e-01],
       [ 8.06958829e-01,  5.51457332e-01],
       [-2.95361869e-01, -4.67210497e-01],
       [-4.79364461e-01,  2.24877018e-01],
       [ 8.62993490e-01, -2.82338227e-01],
```

numpy.random.randn(*d0, d1, ..., dn*)

Return a sample (or samples) from the "standard normal" distribution.

If positive, int_like or int-convertible arguments are provided, randn generates an array of shape `(d0, d1, ..., dn)`,

Question6

(i) Implement the Naïve K-means (the basic/standard algorithm shown in lecture) clustering algorithm to find the 3 cluster centroids. Classify the data based on the three centroids found and illustrate the results using a plot (e.g., mark the 3 clusters of data points using different colours).

(i) SOLUTION

K-Means Clustering::

- Randomly select K centroids. i

Until centroid locations converge / For fixed # of iterations:

For each data sample :

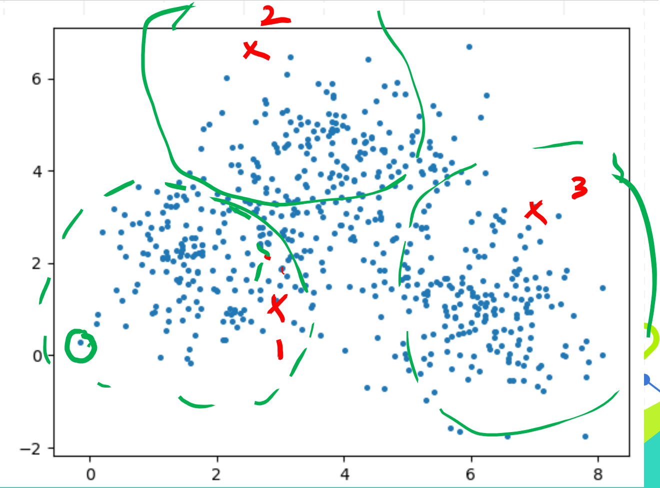
For each centroid : $\Delta x^2 + \Delta y^2$

- Compute Euclidean distance from data point to current centroid
- Assign closest centroid as label

- Calculate average of labelled data and assign as new centroids.
- Repeat above until centroid locations stabilise

Basic K-means Clustering

1. First, you choose K — the number of clusters. Then you randomly put K feature vectors, called **centroids**, to the feature space.
2. Next, compute the distance from each example **x** to each centroid **c** using some metric, like the Euclidean distance. Then we assign the closest centroid to each example (like if we labeled each example with a centroid id as the label).
3. For each centroid, we calculate the average feature vector of the examples labeled with it. These average feature vectors become the new locations of the centroids.
4. We recompute the distance from each example to each centroid, modify the assignment and repeat the procedure until the assignments don't change after the centroid locations are recomputed.
5. Finally we conclude the clustering with a list of assignments of centroids IDs to the examples.



Question6

(i) Implement the Naïve K-means (the basic/standard algorithm shown in lecture) clustering algorithm to find the 3 cluster centroids. Classify the data based on the three centroids found and illustrate the results using a plot (e.g., mark the 3 clusters of data points using different colours).

(i) SOLUTION

K-Means Clustering::

- Randomly select K centroids.

```
##
K=3 # number of clusters
##
##Step 1: Initialize the centroids randomly from the data points:
Centroids=np.array([]).reshape(d,0)
for i in range(K):
    rand=rd.randint(0,m-1) # randomly pick a number from 0 to m-1
    Centroids=np.c_[Centroids,X[rand]] #concatenation along the second axis
Output={}
```

```
500
[[7.02681899]
 [0.82485523]]
299
[[7.02681899 4.45080345]
 [0.82485523 5.3726352 ]]
564
[[7.02681899 4.45080345 6.86306252]
 [0.82485523 5.3726352  1.30328999]]
```

numpy.c_

numpy.c_ = *<numpy.lib.index_tricks.CClass object>*

Translates slice objects to concatenation along the second axis.

```
>>> np.c_[np.array([1,2,3]), np.array([4,5,6])]
array([[1, 4],
       [2, 5],
       [3, 6]])
```

```
print(Centroids[:,0]) [7.02681899 0.82485523]
print(Centroids[:,1]) [4.45080345 5.3726352 ]
print(Centroids[:,2]) [6.86306252 1.30328999]
```

Question6

(i) Implement the Naïve K-means (the basic/standard algorithm shown in lecture) clustering algorithm to find the 3 cluster centroids. Classify the data based on the three centroids found and illustrate the results using a plot (e.g., mark the 3 clusters of data points using different colours).

(i) SOLUTION

100

```
##Repeat step 2 till n_iter/convergence is achieved.
for i in range(n_iter):
    #Step 2.a: For each training example compute the euclidian distance from the
    EuclidianDistance=np.array([]).reshape(m,0)

    for k in range(K):
        # Compute the distance between the kth centroid and every data point
        tempDist=np.sum((X-Centroids[:,k])**2,axis=1)

        # stack the K sets of Euclid distance in K columns
        EuclidianDistance=np.c_[EuclidianDistance,tempDist]

    # Center indicator: locate the column (argmin) that has the minimum distance
    C=np.argmin(EuclidianDistance,axis=1)+1
```

$\Delta x^2 + \Delta y^2$ (for each pt.)

e.g. k=0

```
>>> ((X-Centroids[:,k])**2)
array([[12.67707731,  0.63767422],
       [ 7.77587121,  1.49391152],
       [ 3.30824086,  6.68865701],
       ...,
       [ 8.0346248 , 21.05017838],
       [ 5.5162205 , 13.79118345],
       [ 1.7156753 , 11.73419003]])
>>> np.sum((X-Centroids[:,k])**2,axis=1)
array([1.33147515e+01, 9.26978274e+00, 9.99689786e+00,
       7.35371374e+00, 1.47469717e+01, 1.12850806e+01,
```

for first centroid

Distance of first point in X from

1st Centroid   2nd Centroid   3rd Centroid

```
>>> EuclidianDistance
array([[36.4098828 ,  4.5246032 , 13.31475153],
       [26.61394776,  1.65939499,  9.26978274],
       [14.88505403,  0.1540912 ,  9.99689786],
```

0   1   2

```
>>> np.argmin(EuclidianDistance,axis=1)
array([1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

Question6

(i) Implement the Naïve K-means (the basic/standard algorithm shown in lecture) clustering algorithm to find the 3 cluster centroids. Classify the data based on the three centroids found and illustrate the results using a plot (e.g., mark the 3 clusters of data points using different colours).

(i) SOLUTION

```
#Step 2.b: We need to regroup the data points based on the cluster index C and store in
#the Output dictionary and also compute the mean of separated clusters and assign it as
#new centroids. Y is a temporary dictionary which stores the solution for one particular iteration.

Y={}
for k in range(K):
    # each Y[k]: array([], shape=(2, 0), dtype=float64)
    Y[k+1]=np.array([]).reshape(d,0)
for i in range(m):
    # Indicate and collect data X according to the Center indicator
    Y[C[i]]=np.c_[Y[C[i]],X[i]] #np.shape(Y[k])=(2, number of points nearest to kth center)
for k in range(K):
    Y[k+1]=Y[k+1].T # transpose the row-wise data to column-wise

# Compute new centroids
for k in range(K):
    Centroids[:,k]=np.mean(Y[k+1],axis=0)

    (Centroids[:,0])
    (Centroids[:,1])
    (Centroids[:,2])
```

Append coordinates of points in this group

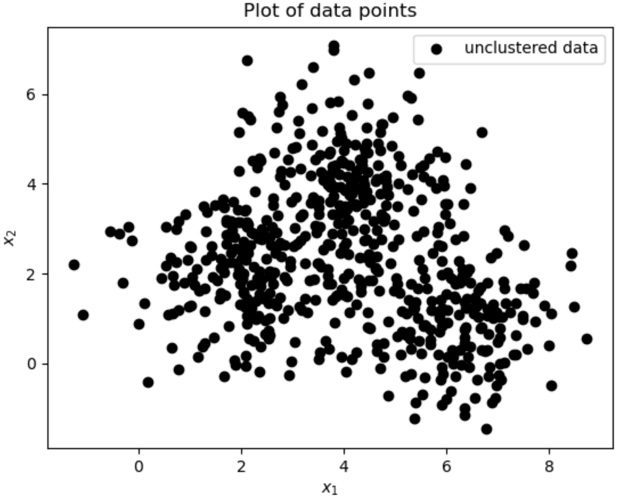
Stores coordinates of all the points in this group

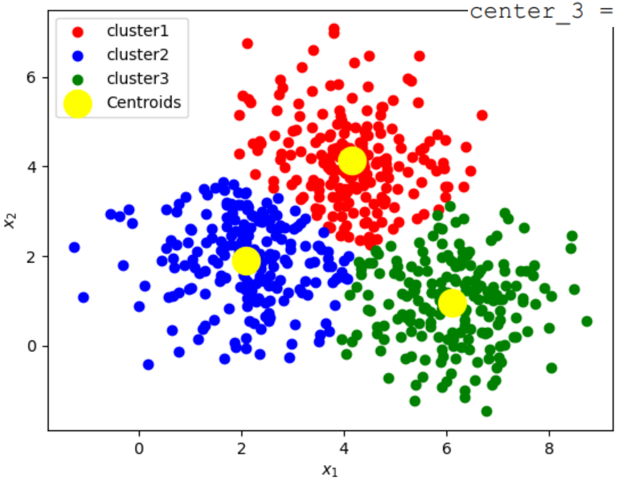
```
>>> Y[1]
array([[ 4.10153110e+00,  1.74235766e+00],
       [ 4.35253253e+00,  1.43172030e+00],
       [ 3.99438212e+00,  9.56512024e-01],
       [ 4.48950249e+00,  2.03746779e+00],
       [ 3.87092738e+00,  2.61452114e-02],
```

```python
## Plot data
plt.scatter(X[:,0],X[:,1],c='black',label='unclustered data')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend()
plt.title('Plot of data points')
plt.show()
## plot clusters
color=['red','blue','green','cyan','magenta']
labels=['cluster1','cluster2','cluster3','cluster4','cluster5']
for k in range(K):
    plt.scatter(Output[k+1][:,0],Output[k+1][:,1],c=color[k],label=labels[k])
plt.scatter(Centroids[0,:],Centroids[1,:],s=300,c='yellow',label='Centroids')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend()
plt.show()
```

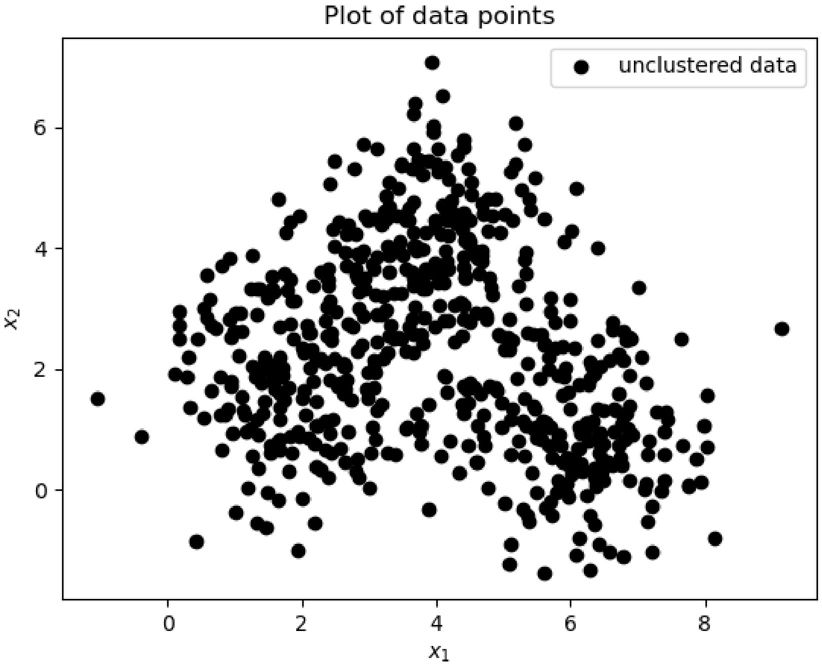
```
>>> firstcentroid.T
array([[6.68483533, 5.14812158],
       [6.68483533, 5.14812158],
       [6.20046256, 4.43510273],
       [5.23128709, 4.31418943],
       [4.56572234, 4.27050583],
       [4.30253491, 4.22216623],
       [4.20347162, 4.15516998],
       [4.18173274, 4.14596907],
       [4.17395956, 4.14292448],
       [4.1645098 , 4.14137727],
       [4.15502922, 4.14652715],
       [4.15502922, 4.14652715],
       [4.15502922, 4.14652715],
       [4.15502922, 4.14652715],
       [4.15502922, 4.14652715],
       [4.15502922, 4.14652715],
       [4.15502922, 4.14652715],
       [4.15502922, 4.14652715],
       [4.15502922, 4.14652715],
       [4.15502922, 4.14652715],
       [4.15502922, 4.14652715],
       [4.15502922, 4.14652715],
```

```python
center_1 = np.array([2,2])
center_2 = np.array([4,4])
center_3 = np.array([6,1])
```

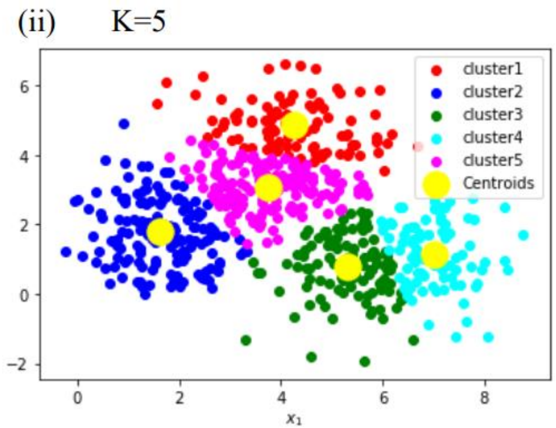




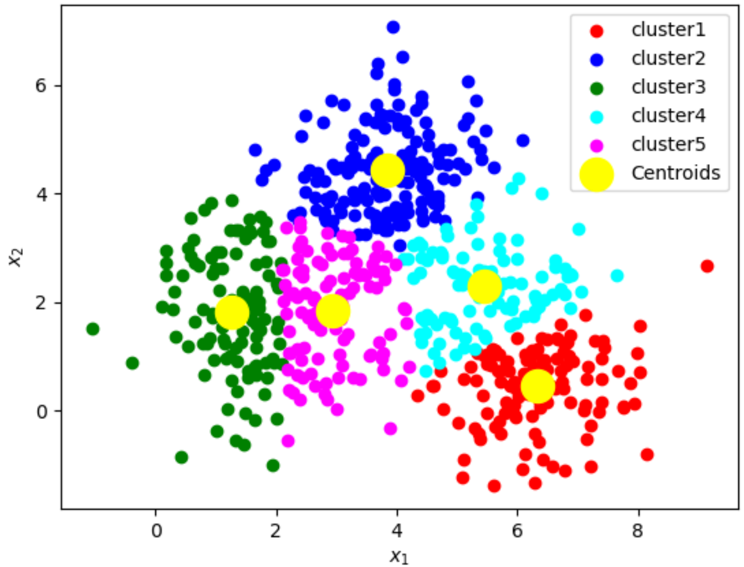
k=5



Plot of data points
unclustered data
6
4
2
0
0
2
4
6
8
x_2
x_1



(ii) K=5
cluster1
cluster2
cluster3
cluster4
cluster5
Centroids
6
4
2
0
−2
0
2
4
6
8
x_1



cluster1
cluster2
cluster3
cluster4
cluster5
Centroids
6
4
2
0
0
2
4
6
8
x_2
x_1

Question 7

Load the iris data "from sklearn.datasets import load_iris". Assume that the class labels are not given. Use the Naïve K-means clustering algorithm to group all the data based on K=3. How accurate is the result of clustering comparing with the known labels?

K-Means Clustering::   3

- Randomly select K centroids.

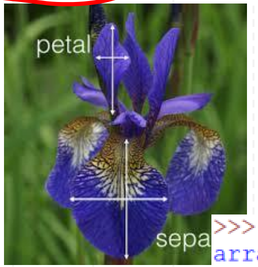
Until centroid locations converge / For fixed # of iterations:

    For each data sample :

        For each centroid :

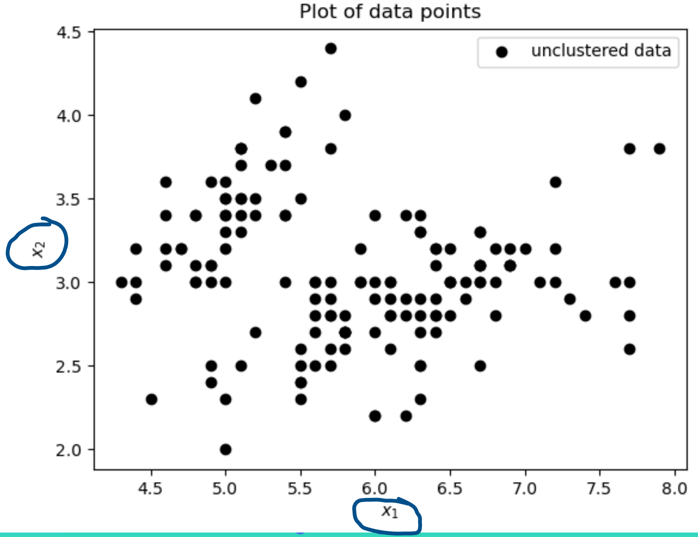
    - Compute Euclidean distance from data point to current centroid
    - Assign closest centroid as label

- Calculate average of labelled data and assign as new centroids.
- Repeat above until centroid locations stabilise




```
>>> X
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
```

```
>>> X[:,0]
array([5.1, 4.9, 4.7, 4.6, 5. , 5.4, 4.6,
       4.3, 5.8, 5.7, 5.4, 5.1, 5.7, 5.1,
       5. , 5.2, 5.2, 4.7, 4.8, 5.4, 5.2,
       5.1, 5. , 4.5, 4.4, 5. , 5.1, 4.8,
```



```
##Step 1: Initialize the centroids randomly from the data points:
Centroids=np.array([]).reshape(d,0)
for i in range(K):
    Xperclass = X[y==i]
    random.seed(a=0)
    rand=random.randint(0,len(Xperclass)) # pick any of the samples from each class for initialization
    Centroids=np.c_[Centroids,Xperclass[rand]]
```

y = 0, 1, 2

```
>>> Xperclass
array([[6.3, 3.3, 6. , 2.5],
       [5.8, 2.7, 5.1, 1.9],
       [7.1, 3. , 5.9, 2.1],
       [6.3, 2.9, 5.6, 1.8],
       [6.5, 3. , 5.8, 2.2],
```

The 50 data samples from one class

```
>>> Centroids
array([[5.006     , 5.9016129 , 6.85      ],
       [3.428     , 2.7483871 , 3.07368421],
       [1.462     , 4.39354839, 5.74210526],
       [0.246     , 1.43387097, 2.07105263]])
```

```
##Step 2.a: For each training example compute the euclidian distance
#from the centroid and assign the cluster based on the minimal distance
Output={}
EuclidianDistance=np.array([]).reshape(m,0)
for k in range(K):
       tempDist=np.sum((X-Centroids[:,k])**2,axis=1)
       EuclidianDistance=np.c_[EuclidianDistance,tempDist
C=np.argmin(EuclidianDistance,axis=1)+1
```

```
>>> X-Centroids[:,k]
array([[-1.75      ,  0.42631579, -4.34210526, -1.87105263],
       [-1.95      , -0.07368421, -4.34210526, -1.87105263],
       [-2.15      ,  0.12631579, -4.44210526, -1.87105263],
       [-2.25      ,  0.02631579, -4.24210526, -1.87105263],
       [-1.85      ,  0.52631579, -4.34210526, -1.87105263],
       [-1.45      ,  0.82631579, -4.04210526, -1.67105263],
```

```
>>> EuclidianDistance
array([[1.99800000e-02, 1.16912747e+01, 2.55989612e+01],
       [2.00380000e-01, 1.15503070e+01, 2.61626454e+01],
       [1.73980000e-01, 1.27403070e+01, 2.78715928e+01],
```

```
>>> C
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 3, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3,
```

```python
##Step 2.b: We need to regroup the data points based on the cluster index C
#and store in the Output dictionary and also compute the mean of separated
#clusters and assign it as new centroids. Y is a temporary dictionary which
#stores the solution for one particular iteration.

Y={}
for k in range(K):
    Y[k+1]=np.array([]).reshape(d,0)
for i in range(m):
    Y[C[i]]=np.c_[Y[C[i]],X[i]]
for k in range(K):
    Y[k+1]=Y[k+1].T

for k in range(K):
     Centroids[:,k]=np.mean(Y[k+1],axis=0)
```

```
>>> Y[2]
array([[7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
```

```
##Repeat step 2 till convergence is achieved.
for i in range(n_iter):
     #step 2.a
      EuclidianDistance=np.array([]).reshape(m,0)
      for k in range(K):
          tempDist=np.sum((X-Centroids[:,k])**2,axis=1)
          EuclidianDistance=np.c_[EuclidianDistance,tempDist]
      C=np.argmin(EuclidianDistance,axis=1)+1
     #step 2.b
      Y={}
      Idx1=[]
      Idx2=[]
      Idx3=[]
      for k in range(K):
          Y[k+1]=np.array([]).reshape(d,0)
      for i in range(m):
          # put X to each cluster vector Y
          Y[C[i]]=np.c_[Y[C[i]],X[i]]
          # collect indices of each clustered group
          if C[i]==1: Idx1 += [i]
          if C[i]==2: Idx2 += [i]
          if C[i]==3: Idx3 += [i]

      for k in range(K):
          Y[k+1]=Y[k+1].T

      for k in range(K):
          Centroids[:,k]=np.mean(Y[k+1],axis=0)
```

```
>>> Y[2]
array([[7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
```

```
>>> Idx1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
2, 23, 24, 25, 26, 27, 28, 29, 30, 31,
2, 43, 44, 45, 46, 47, 48, 49]
>>> Idx2
[50, 51, 53, 54, 55, 56, 57, 58, 59, 6
 71, 72, 73, 74, 75, 76, 78, 79, 80, 8
 92, 93, 94, 95, 96, 97, 98, 99, 101,
3, 138, 142, 146, 149]
```

index of pts. in cluster 1

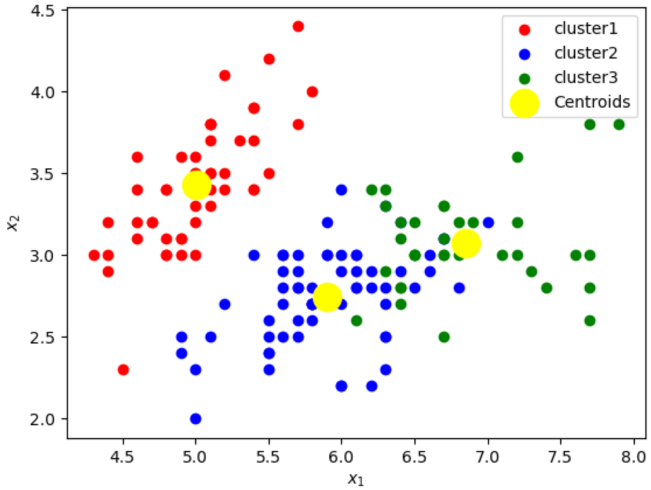
cluster 2

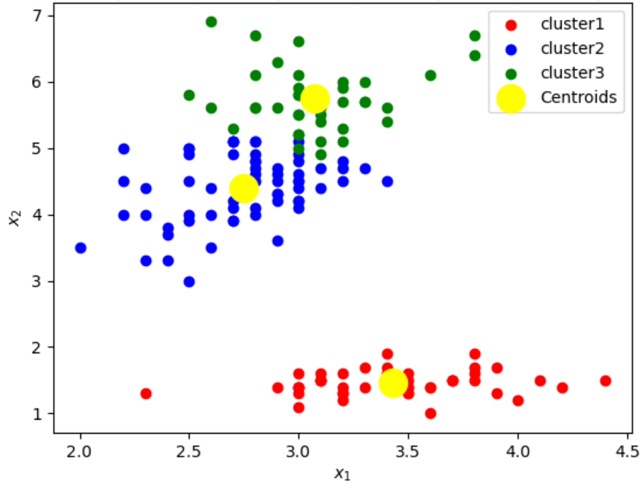
```python
## Compute classification accuracy
Class1_est = y[Idx1]
Class2_est = y[Idx2]
Class3_est = y[Idx3]
Cls1_correct = 0
for i in range(len(Class1_est)):
    if Class1_est[i]==0: Cls1_correct +=1
Cls2_correct = 0
for i in range(len(Class2_est)):
    if Class2_est[i]==1: Cls2_correct +=1
Cls3_correct = 0
for i in range(len(Class3_est)):
    if Class3_est[i]==2: Cls3_correct +=1
Accuracy = (Cls1_correct+Cls2_correct+Cls3_correct)/len(y)
print(Accuracy)
```

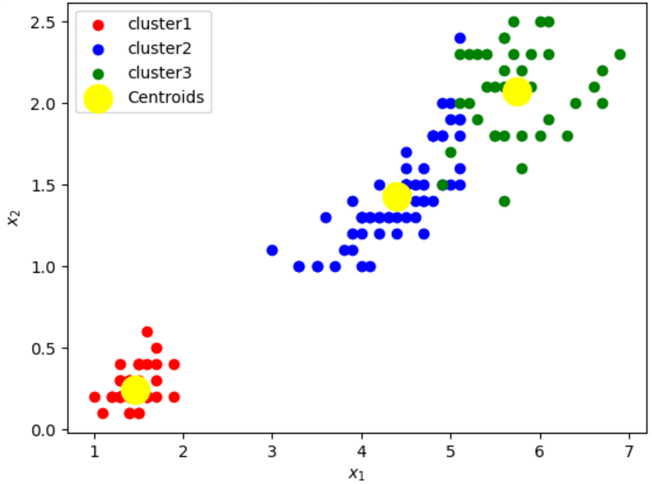
→ actual class of those pts. in cluster 1 (value = 0)

```
>>> Class2_est
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
       1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2
```

```
0.8933333333333333
```





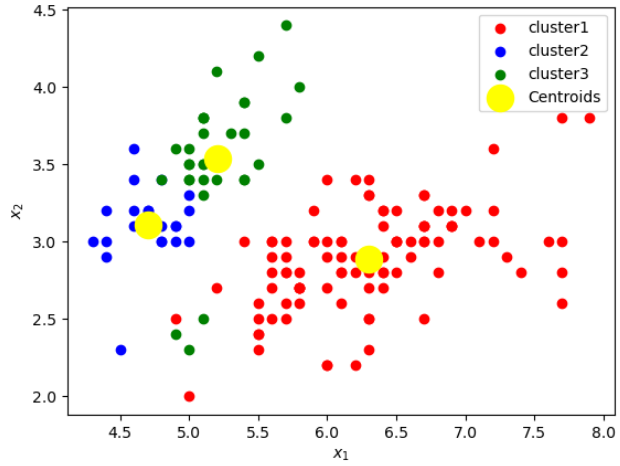


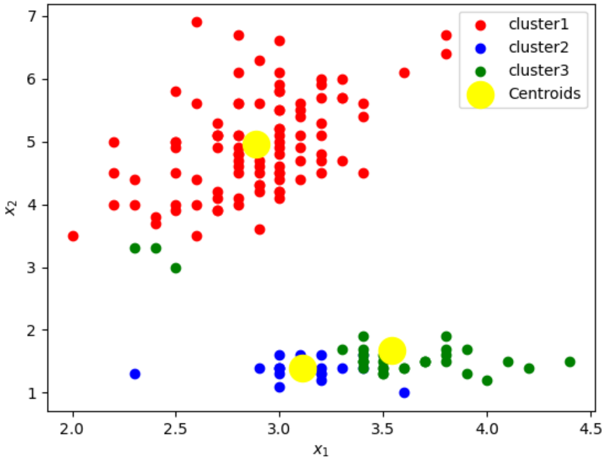
If initialization is different...

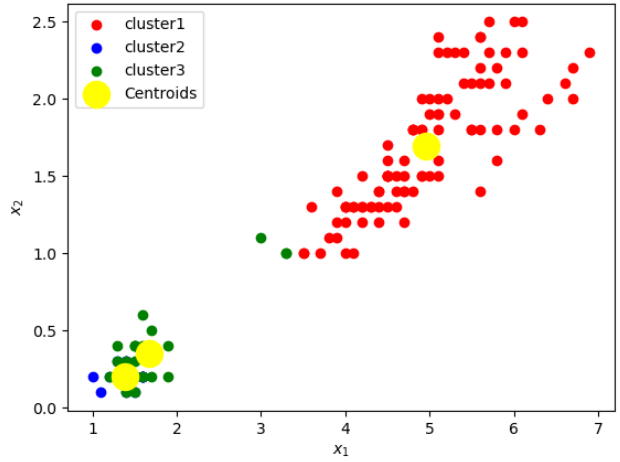
Centroids

```
[[6.9]
 [3.1]
 [4.9]
 [1.5]]
[[6.9 4.6]
 [3.1 3.6]
 [4.9 1. ]
 [1.5 0.2]]
[[6.9 4.6 5.4]
 [3.1 3.6 3.9]
 [4.9 1.  1.3]
 [1.5 0.2 0.4]]
[[6.30103093 4.725      5.23793103]
 [2.88659794 3.15416667 3.54827586]
 [4.95876289 1.41666667 1.67931034]
 [1.69587629 0.19583333 0.36896552]]
[[6.30103093 4.70909091 5.21612903]
 [2.88659794 3.13181818 3.53870968]
 [4.95876289 1.39090909 1.68064516]
 [1.69587629 0.19545455 0.35806452]]
[[6.30103093 4.7        5.20625   ]
 [2.88659794 3.10952381 3.540625  ]
 [4.95876289 1.39047619 1.671875  ]
 [1.69587629 0.2        0.35      ]]
[[6.30103093 4.7        5.20625   ]
 [2.88659794 3.10952381 3.540625  ]
 [4.95876289 1.39047619 1.671875  ]
 [1.69587629 0.2        0.35      ]]
[[6.30103093 4.7        5.20625   ]
 [2.88659794 3.10952381 3.540625  ]
 [4.95876289 1.39047619 1.671875  ]
 [1.69587629 0.2        0.35      ]]
```

```
rand=random.randint(0,len(X))
Centroids=np.c_[Centroids,X[rand]]
print(Centroids)
```







The K-means clustering method uses the target labels for calculating the distances from the cluster centroids for clustering.

True

False

NUS National University of Singapore

The fuzzy C-means algorithm groups the data items such that an item can exist in multiple clusters.

True

False

When poll is active, respond at **pollev.com/cdj**

Text **CDJ** to **+61 480 025 509** once to join

How can you prevent a clustering algorithm from getting stuck in bad local optima?

- Set the same seed value for each run
- Use the bottom ranked samples for initialization
- Use the top ranked samples for initialization
- All the above
- None of the above

When poll is active, respond at **pollev.com/cdj**

Text **CDJ** to **+61 480 025 509** once to join

Consider the following data points: x=[1,1], y=[0,1] and z=[0,0]. The k-means algorithm is initialized with centers at x and y. Upon convergence, the two centres will be at

- x and z
- x and y
- y and the midpoint of y and z
- z and the midpoint of x and y
- None of the above

Consider the following 8 data points... The k-means algorithm is initialized with centers at [0,0] and [3,0]. The first center after convergence is c_1=[0.5,0.5]. The second centre after convergence is c_2 is __, __.