Assignment 1

Name: Ipsa Mishra
Student ID: 1001759616

Questions:

1. Write a function cluster = mykmeans(X, k) that clusters data X into k clusters. Use l_2- norm of vectors to compute distance between different samples

```
def mykmeans(X,k):
    n_iter = 200
    n_X = np.shape(X)[0]
    tags = np.zeros(n_X)

    iter_centers = []

    c = np.zeros((k,np.shape(X)[1]))
    for i in range(0,k):
        c[i] = 5 * np.random.rand(1,np.shape(X)[1])

    for nc in range(len(c)):
        iter_centers.append([])

    for nc in range(len(c)):
        iter_centers[nc].append(list(c[nc]))

    for i in range(n_iter):
        for j in range(n_X):
            dist = [np.linalg.norm(X[j] - c[n]) for n in range(len(c))]
            indx = np.where(dist == np.amin(dist))
            indx = list(indx[0])
            tags[j] = indx[0]

    upd_c = c * 0.0
    for j in range(len(c)):
        identify = (tags == j) + 0.0
        upd_c[j] = np.matmul(np.transpose(identify) , X) / np.sum(identify)

    c = upd_c
    for nc in range(len(c)):
        iter_centers[nc].append(list(c[nc]))

    iter_centers = np.array(iter_centers)

    for nc in range(len(c)):
        plt.scatter(iter_centers[nc,:,0],iter_centers[nc,:,1],marker = '.')

    print('# of iterations = ', len(iter_centers[0])-1)
    return c, tags
```

2. Generate X using the parameters above, and you should know the ground truth for cluster centers and cluster assignments. (You need to show data from 3 different clusters simultaneously.) Generate N = 300 for each cluster and test your implementation on the synthetic data with different k = 2, 3, 4, 5. Visualize the changes of clustering result and cluster centers, compute clustering accuracy when k = 3. How accurate are the estimated cluster centers when k = 3?

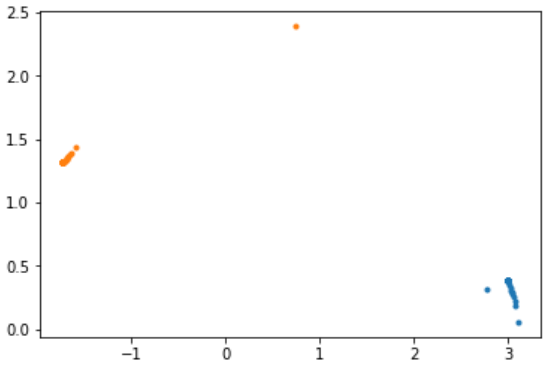
```
def generate_data(mu1,mu2,mu3,sigma,n_samples):

    x1 = np.random.multivariate_normal(mu1,sigma,n_samples)
    x2 = np.random.multivariate_normal(mu2,sigma,n_samples)
    x3 = np.random.multivariate_normal(mu3,sigma,n_samples)
    x = np.concatenate((x1,x2,x3),axis=0)
    return x
```

Here n_samples = 300 , mu1 = [-3,0], mu2 = [3,0], mu3 = [0,3], sigma = [[1,0.75],[0.75,1]]

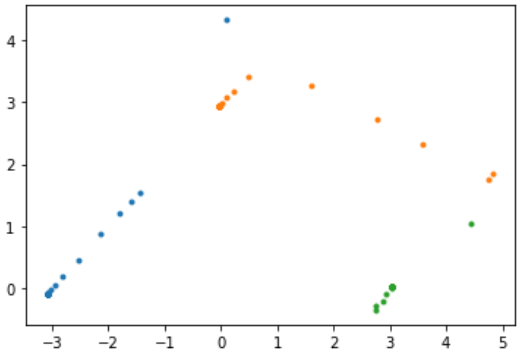
When k = 2:

```
# of iterations =  200
[[ 2.99587617  0.38030594]
 [-1.72442853  1.31286611]]
```



When k = 3:

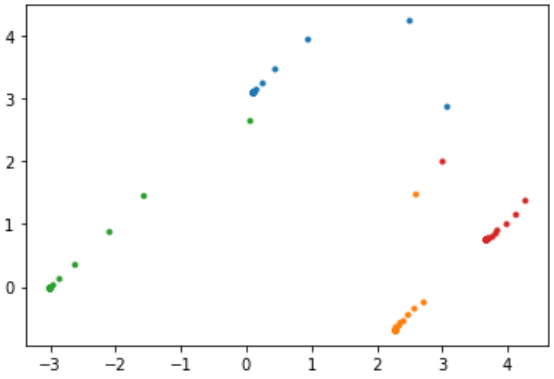
```
# of iterations =  200
[[-3.08395337 -0.08533732]
 [-0.02371148  2.94511067]
 [ 3.04268933  0.02065249]]
```



When k = 4:

# of iterations = 200

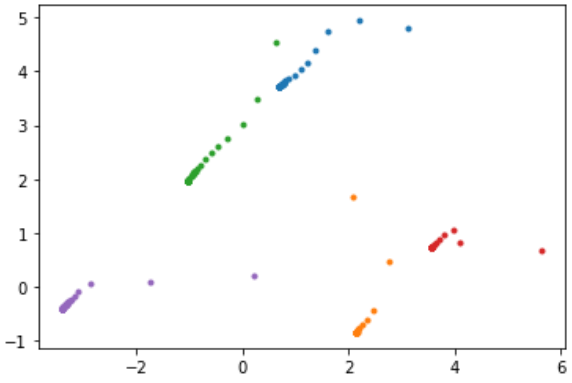
[[0.09853316 3.11266869]
[2.2646721 -0.68223138]
[-3.02168295 -0.01659694]
[3.67043944 0.77438305]]



When k = 5

# of iterations = 200

[[-0.46761013 2.56022865]
[3.12543079 0.09587834]
[-3.63252164 -0.71638275]
[1.00666164 3.88715703]
[-2.14912655 0.80037338]]



The plots show that we are reaching completion in few iterations.
The error found of the clustering result and cluster centers when k = 3 is 0.07564170692388537. This suggests that it has higher accuracy as the error is quite low. The computation of error result can be found in the file kmeans_accuracy.py

3. Now, change the μ to $\mu_1 = [-2, 0]$, $\mu_2 = [2, 0]$, $\mu_3 = [0, 2]$, generate new X and test your k-means algorithm. Is this X easier or harder than the previous case? Generate N = 300 for each cluster and test your implementation on the synthetic data with different k = 2, 3, 4, 5. Visualize the changes of clustering result and cluster centers, compute clustering accuracy when k = 3. How accurate are the estimated cluster centers when k = 3?

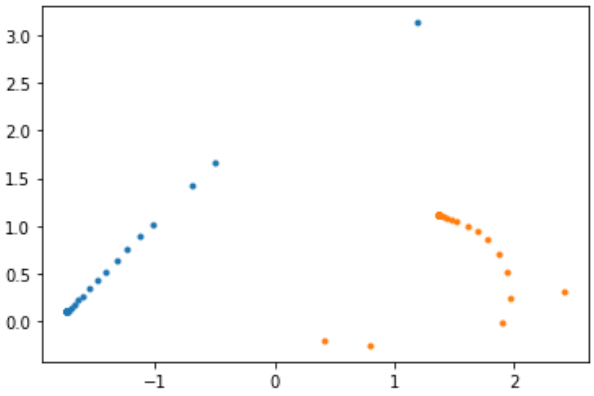
X is harder here as the true center values are closer to one another which makes it harder to distinguish between different classes which can be known from the following plots.

When k = 2:

# of iterations = 200

[[-1.73707729 0.10201998]

[1.36752722 1.11480949]]



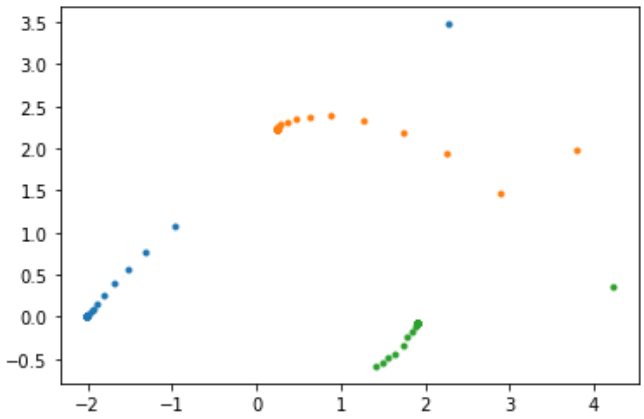
When k = 3:

# of iterations = 200

[[-2.01727152 0.00628048]

[0.23631406 2.23623991]

[1.91298001 -0.07510182]]



When k = 4:

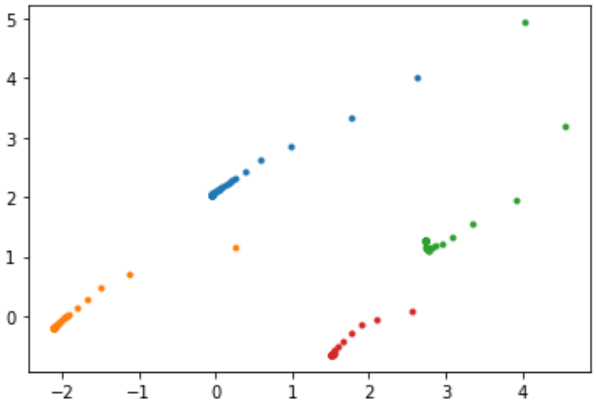
# of iterations = 200

[[-0.04313196 2.04785013]

[-2.11309275 -0.19051898]

[2.73964177 1.27468126]

[1.5369152 -0.63229772]]



When k = 5:

# of iterations = 200

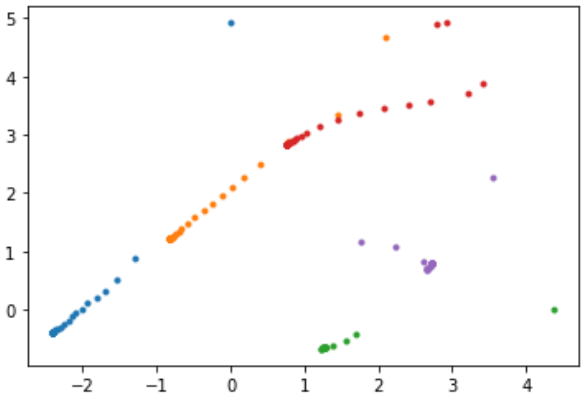
[[-2.41656514 -0.40357043]

[-0.81656084 1.20705512]

[1.27050755 -0.64547016]

[0.76768159 2.84556238]

[2.72265892 0.79687719]]



The error in this case is 0.15582362375539718 which suggests that the estimated clustering centers here are less accurate than the estimated clustering centers when k = 3.

Question 2:

1. Write a function class = myknnclassify(train, test, k) that classifies the class of input test given a training set train using k-NN classifier where k is the number of neighbors. Use the following parameters to generate 2D Gaussian random samples for different clusters. Try out different k = 1, 2, 3, 4, 5, 10, 20 to test your k-NN classifier. Show the changes of accuracy w.r.t. the k.

```
def myknnclassify(train, test, k):

    d = np.shape(test)[1] - 1

    n_test = len(test)

    n_train = len(train)

     p_test = np.zeros(n_test)

     for i in range(n_test):

         dist = np.zeros(n_train)

         for j in range(n_train):

             dist[j] = np.linalg.norm(train[j,0:d] - test[i,0:d])

         classify = train[dist.argsort()[:k],d]

          p_test[i] = np.round(np.mean(classify))

return p_test
```

Changes of accuracy w.r.t k:

For k = 1 Accuracy is 0.81

For k = 2 Accuracy is 0.84

For k = 3 Accuracy is 0.82

For k = 4 Accuracy is 0.85

For k = 5 Accuracy is 0.86

For k = 10 Accuracy is 0.84

For k = 20 Accuracy is 0.87

Here higher the accuracy, the more accurate is the result.

2. Write a function value = myknnregress(X, test, k) that that regresses the target value of input test given a training set train using k-NN regressor where k is the number of neighbors. Try out different k = 1,2,3,5,10,20,50,100 to test your k-NN classifier. Show the changes of accuracy in average l_2-norm w.r.t. the k.

```
def myknnregress(train, test, k):
    d = np.shape(test)[1] - 1
    n_test = len(test)
    n_train = len(train)
    p_test = np.zeros(n_test)
    for i in range(n_test):
        dist = np.zeros(n_train)
        for j in range(n_train):
            dist[j] = np.linalg.norm(train[j,0:d] - test[i,0:d])
        regress = train[dist.argsort()[:k],d]
        p_test[i] = np.average(regress, weights = 1 / dist[dist.argsort()[-k:]])
    return p_test
```

Changes of accuracy w.r.t k:

For k = 1 Error is 6.965926445750244

For k = 2 Error is 6.283326093893768

For k = 3 Error is 6.0152352706637195

For k = 5 Error is 5.792306708849333

For k = 10 Error is 5.646513446613116

For k = 20 Error is 5.577313293164228

For k = 50 Error is 6.422481335744632

For k = 100 Error is 8.378160169602335

Lower error means more accuracy. Thus lower the error, higher the accuracy.

3. Write a function for locally weighted regression [value weight] = myLWR(X, test, k) that classifies the class of an input test given a training set X using k-NN classifier where k is the number of neighbors. Use l_2-norm of vectors to compute distance between different samples. Try out different k = 1, 2, 3, 5, 10, 20,50,100 to test your k-NN classifier. Show the changes of accuracy in average l_2-norm w.r.t. the k.

```
def myLWR(train, test, k):

    d = np.shape(test)[1] - 1

    n_test = len(test)

    n_train = len(train)

    p_test = np.zeros(n_test)

    y_test = test[:,d]

   X = np.zeros((len(test),k))

  for i in range(n_test):

      dist = np.zeros(n_train)

      for j in range(n_train):

            dist[j] = np.linalg.norm(train[j,0:d] - test[i,0:d])
```

```
            X[i,:] = train[dist.argsort()[:k],d]
        weight1 = np.linalg.inv(np.matmul(np.transpose(X),X))
         weight2 = np.matmul(np.transpose(X),y_test)
        weight = np.matmul(weight1,weight2)
        p_test = np.matmul(X,weight)
         return p_test, weight
      For k =  1  Error is  7.880317252052194
       For k =  1  Weights are  [0.93435996]
       For k =  2  Error is  6.615640000340855
       For k =  2  Weights are  [0.43144128 0.55129728]
      For k =  3  Error is  6.532519345167462
     For k =  3  Weights are  [0.38414984 0.46133077 0.1499889 ]
     For k =  5  Error is  6.303354374807807
     For k =  5  Weights are  [0.30046239 0.35459489 0.05543681 0.20250969 0.09821573]
    For k =  10  Error is  5.97068682493843
    For k =  10  Weights are  [ 0.24613186  0.30221063 -0.00944793  0.1331363   0.03136115  0.23223719
     0.15646315 -0.0048481  -0.05651537 -0.00907378]
 For k =  20  Error is  5.64735999246705
 For k =  20  Weights are  [ 2.07442895e-01  2.93032627e-01 -8.85856777e-05  1.19303171e-01
-2.26657471e-02  1.55334410e-01  1.31447338e-01 -5.95311768e-02
-1.43427255e-01 -1.51470158e-02  1.24684323e-01 -6.49027781e-03
-3.85624501e-02  3.13432604e-02  4.35241400e-02  1.12364489e-01
 8.55746564e-02  1.12461746e-02 -1.15724059e-02  3.19852482e-02]
```

For k = 50 Error is 4.332256755051735

For k = 50 Weights are [0.16482149 0.10793502 0.01330409 0.20934604 -0.04300375 0.03708644
0.15928762 0.07399963 -0.36608011 -0.04052797 0.11037056 0.03316662
0.13842793 0.01622619 0.04825703 0.17709621 0.11214667 -0.0615732
-0.03363287 0.04909454 0.15937603 0.04415691 0.18490905 0.13266618
0.04520929 0.01033449 0.02409583 -0.03975858 0.02762938 0.13480803
0.02184269 -0.07686584 0.10072722 0.08376757 -0.14373192 0.0362371
-0.05741445 -0.00425124 -0.12425525 -0.06339309 0.06561517 -0.1219339
-0.10761755 0.02753932 0.05668823 -0.14102981 -0.00898446 -0.11571499
0.043171 -0.05358787]

For k = 100 Error is 2.791858434333119e-09

For k = 100 Weights are [-0.28131865 0.41952539 0.20327404 -0.77878338 0.056476 0.86879814
-0.64680983 -0.55206047 0.08984684 0.16714951 -0.07226204 0.26063284
-0.02812533 0.17969412 -0.39377466 -0.07577328 0.34004063 0.42178499
0.25382739 0.68591251 0.28906124 -0.50141323 -0.38597571 -0.08399256
0.24865924 0.56202627 0.44558627 0.18394641 0.6870592 -0.00993417
0.64281417 0.16066303 0.16789928 0.03662411 -0.10304432 0.08758792
-0.25357511 -0.53256211 -0.29054675 0.53124045 -0.18202999 -0.01949241
-0.05560941 0.13091118 -0.3230798 -0.17888693 -0.31019655 -0.02298704
-0.31641683 -0.05452224 0.08198529 0.00363478 -0.05124903 -0.12134997
-0.0300232 -0.12850794 -0.11277869 -0.0404284 -0.0172647 0.00646658

0.06731841 -0.26636761 -0.1799334 0.10779231 -0.38783716 -0.23769239

0.03005949 0.15806486 -0.11313388 0.1319522 -0.12164654 0.03990632

-0.1710966 0.4858388 0.30896914 0.0313703 0.16401482 -0.0411818

-0.71409076 0.1301714 -0.22171498 0.11560417 0.33198195 0.67003144

-0.29218687 -0.05110391 0.27110197 -0.07827495 0.00879934 0.40121141

-0.13268525 0.22641832 -0.34997446 0.30484527 -0.51659897 0.32387539

0.07431125 -0.54123175 -0.14776251 -0.09509737]

Here I have calculated the best set of weights which gives the best regression value for the test inputs.