Assignment 1

Zahidur Talukder
Student ID- 1001771748

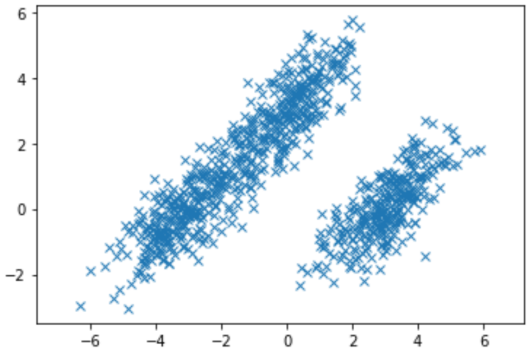
Problem 1.1

The main requirement of this problem is to write a function cluster = mykmeans(X, k) that clusters data X into k clusters. The function is attached here,

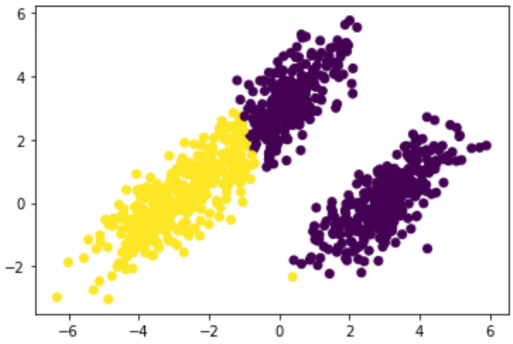
```
## Code begins here

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

def mykmeans(X, k):
    data=X
    N = data.shape[0]
    numFeatures = data.shape[1]
    labels = -1*np.zeros(N)
    numIterationsCompleted = 0
    means = np.zeros((k,numFeatures))

    min = np.min(data)
    max = np.max(data)

    #Fill mean vector with random values(initial mean)
    for i in range(k):
        for f in range(numFeatures):
            means[i][f] = random.uniform(min, max)

     # Implementing l2 norm to get mean distances
    while(True):
        for i in range(N):
            distmin =np.sqrt((np.linalg.norm(data[i] - means[0]))**2)
            minindex = 0
            for j in range(k):
                #Find distance to each mean. Take the smallest one.
                distcurrent = np.sqrt((np.linalg.norm(data[i] - means[j
    ]))**2)
                if(distcurrent < distmin):
                    distmin = distcurrent
                    minindex = j
            labels[i] = minindex

        dummy = True
        for i in range(k):
            datalabeli = data[np.where(labels == i)]
            newmeanforlabeli = np.mean(datalabeli,0)
            for f in range(numFeatures):
                dummy = dummy and (means[i][f] == newmeanforlabeli[f])
        if(dummy == True):
            #Optimum Found!
            return labels, means
        else:
            for i in range(k):
                datalabeli = data[np.where(labels == i)]
                newmeanforlabeli = np.mean(datalabeli,0)
                means[i] = newmeanforlabeli
        numIterationsCompleted+=1

```

Problem 1.2

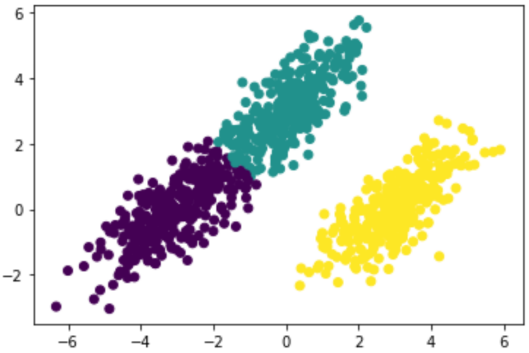
In this problem I have generated 2D Gaussian random samples for 3 different clusters with different means. We know the mean and the co-variance of the data and using our mykmeans function, I generated different clusters for different k values i.e. 1,2,3,4,5.



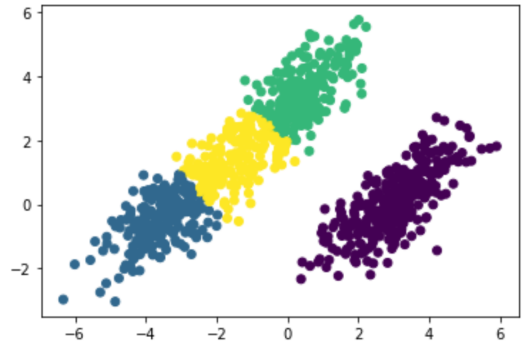
(a) Basic Cluster



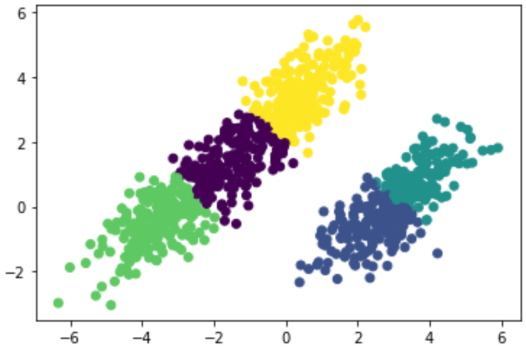
(b) 2 clusters using mykmeans



(c) 3 clusters using mykmeans



(d) 4 clusters using mykmeans



(e) 5 clusters using mykmeans

Figure 1: Clustering Using mykmeans function with different k value

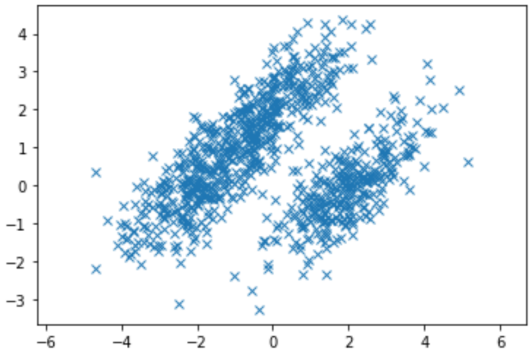
Result evaluation

Here I have successfully computed different clusters with different k values. Now when k = 3 i.e. cluster size is equal to our known cluster, the cluster center is $\begin{pmatrix} -3.10235603 & -0.10392129 \\ 0.07972844 & 3.0767223 \\ 3.04047112 & 0.0821431 \end{pmatrix}$ which is very close to our actual cluster center $\begin{pmatrix} -3 & 0 \\ 0 & 3 \\ 3 & 0 \end{pmatrix}$. We have calculated the accuracy of our model for known cluster. 31 out of 900 points were clustered differently than our ground truth cluster. So the accuracy of our model is 96.5%.

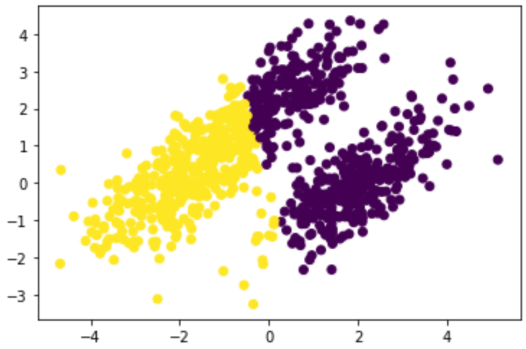
For all other k values, my model can successfully cluster it. In the previous figures we can easily verify my proposition. The cluster center that my model generates for k = 2, 4, 5 are- $\begin{pmatrix} -2.57694328 & 0.42112885 \\ 2.01650278 & 1.51322247 \end{pmatrix}$, $\begin{pmatrix} -3.44428915 & -0.51641228 \\ -1.50037926 & 1.47748377 \\ 3.13452612 & 0.17466947 \\ 0.50953029 & 3.47307962 \end{pmatrix}$, $\begin{pmatrix} 0.50953029 & 3.47307962 \\ -3.46132943 & -0.50573339 \\ -1.50037926 & 1.47748377 \\ 3.85777958 & 0.86183013 \\ 2.40019232 & -0.52232282 \end{pmatrix}$. So, my model can successfully cluster data for any k values.

Problem 1.3

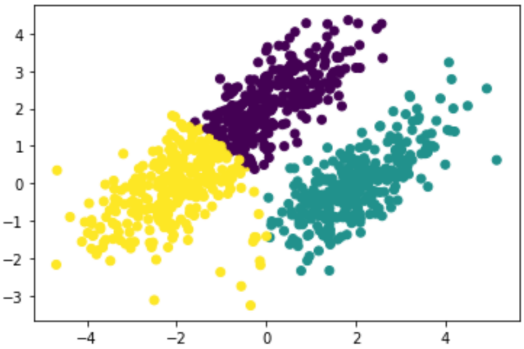
In this problem, i have regenerated 2D Gaussian random samples using new means. Then I have applied mykmeans function again to cluster the new samples with different k values. The result is summarized below.



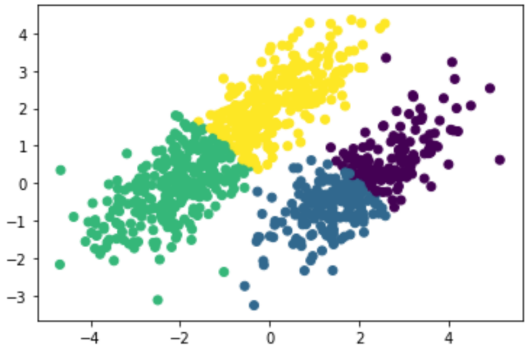
(a) Basic Cluster



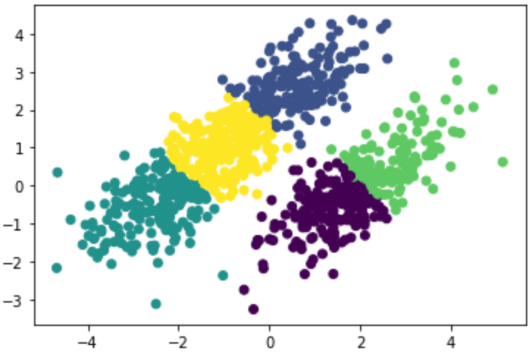
(b) 2 clusters using mykmeans



(c) 3 clusters using mykmeans



(d) 4 clusters using mykmeans



(e) 5 clusters using mykmeans

Figure 2: Clustering Using mykmeans function with different k value

Result evaluation

Here I have successfully computed different clusters with different k values. Now when k = 3 i.e. cluster size is equal to our known cluster, the cluster center is $\begin{pmatrix} 0.11912539 & 2.14876619 \\ 2.03598487 & -0.02135304 \\ -2.06403939 & -0.13991048 \end{pmatrix}$ which is very close to our actual cluster center $\begin{pmatrix} -2 & 0 \\ 0 & 2 \\ 2 & 0 \end{pmatrix}$. We have calculated the accuracy of our model for known cluster. 99 out of 900 points were clustered differently than our ground truth cluster. So the accuracy of our model is 89%.

Here we can see the accuracy is decreased, it is normal since the clusters are now more congested. So this X is harder than the previous case. The graph for clustering with other K values is plotted above. Hence mykmeans function is able to cluster with different k values in this case also.

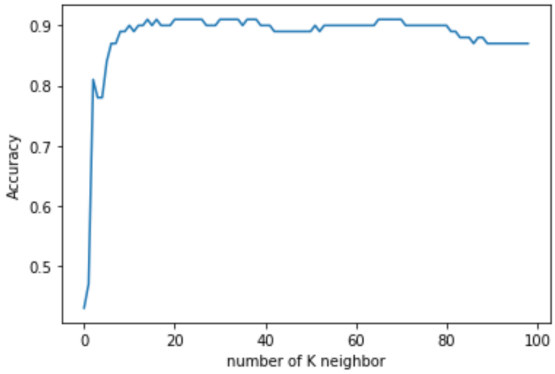
For all other k values, my model can successfully cluster it. In the previous figures we can easily verify my proposition. The cluster center that my model generate for K = 2,,4,5 respectively are- $\begin{pmatrix} 1.44356268 & 1.0003368 \\ -1.72628827 & 0.23360976 \end{pmatrix}$, $\begin{pmatrix} 2.78952771 & 0.72534152 \\ 1.28041752 & -0.71935143 \\ -2.12847055 & -0.0907915 \\ 0.10821226 & 2.13949907 \end{pmatrix}$, $\begin{pmatrix} 1.29647236 & -0.70186629 \\ 0.60706889 & 2.67916345 \\ -2.54664521 & -0.54178034 \\ 2.81672774 & 0.71250915 \\ -0.96209777 & 1.05592762 \end{pmatrix}$

Problem 2.1

The main requirement of this problem is to write a function class = myknnclassify(train, test, k) that classifies the class of input test given a training set train using k-NN classifier wherek K is the number of neighbors. The function is attached here,

```
## Code begins here

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

def myknnclassify(train,test,k):
    estimate_class=[]
    k=k
    test_val=test[:,0:2]
    train_val=train[:,0:2]
    test_len=test_val.shape[0]
    train_len=train_val.shape[0]
    for i in range(test_len):
        distance = []
        for j in range(train_len):
            dist=np.linalg.norm(test_val[i-1]-train_val[j-1])
            distance.append([train[j-1,2],dist])

        # Sort data based on distance
        distance.sort(key=lambda x:x[1])
        zero_class=0;
        one_class=0;

        #Estimate class
        zero_class=0;
        one_class=0;
        for l in range(k):
            distance= np.array(distance)
            if distance[l-1,0]==0:
                zero_class +=1
            elif distance[l-1,0]==1:
                one_class +=1
        if zero_class>=one_class:
            estimate_class.append(0)
        else:
            estimate_class.append(1)


    # finding Accuracy
    number_accurate=0
    for m in range(test_len):
        if test[m-1,2]==estimate_class[m]:
            number_accurate +=1
    accuracy=(number_accurate/test_len)
    print(accuracy,number_accurate,estimate_class)

```

Result evaluation



K Nearest Neighbor (KNN) is a lazy algorithm which determines it's neighbor based on closest distance and then classify its class based on the class of majority neighbor. In this problem, I have formulated the KNN algorithm to check its accuracy with increasing K value. For my understanding in this particular problem, The accuracy is higher with higher k value. I have plotted the accuracy of the model by taking the neighbor value from 0 to 100. The plot of Accuracy Vs Number of Neighbor is attached here.

The required accuracy for different k values is summarize below-

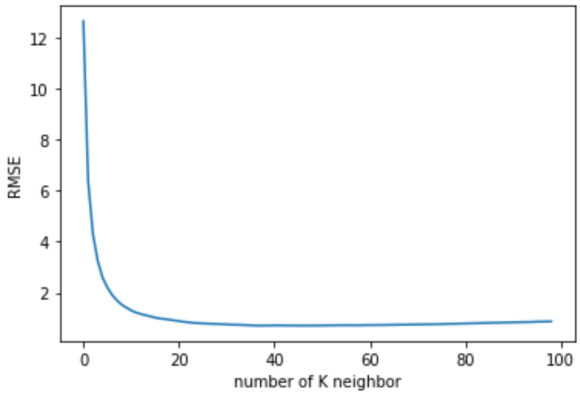
K value	Accuracy
1	0.43
2	0.47
3	0.81
4	0.78
5	0.78
10	0.89
20	0.9

Problem 2.2

The main requirement of this problem is to write a function value = myknnregress(X, test, k) that regresses the target valuenof inputesgiven a training set train using k-NN regressor where k is the number of neighbors. The function is attached here,

```
## Code begins here

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

def myknnregress(X,Test,k):
    estimate_value_table=[]
    k=k
    Train=X
    test=Test[:,0:2]
    train=Train[:,0:2]
    test_len=test.shape[0]
    train_len=train.shape[0]
    for i in range(test_len):
        distance = []
        for j in range(train_len):
            dist=np.linalg.norm(test[i-1]-train[j-1])
            distance.append([Train[j-1,2],dist])
            # Sort data based on distance
        distance.sort(key=lambda x:x[1])


            #Estimate value
        sum_neighbor=0
        for l in range(k):
            distance = np.array(distance)
            sum_neighbor = sum_neighbor+distance[l-1,0]
        estimate_value=sum_neighbor/k
        estimate_value_table.append(estimate_value)

        # finding Accuracy (RMSE)
    sum_rmse=0
    for m in range(test_len):
        sumit=np.square(Test[m-1,2]-estimate_value_table[m])
        sum_rmse=sum_rmse+sumit
    rmse=np.sqrt(sum_rmse/test_len)
    print(rmse)

```

Result evaluation



K Nearest Neighbor (KNN) regression is a lazy algorithm which determines it's neighbor based on closest distance and then estimate it's value based on the mean of value of all the neighbor. In this problem, I have formulated the KNN regression algorithm to check its RMSE with increasing K value. We know RMSE is a universal approch to determine the accuracy of any regression model. If the RMSE is low, the model is more accurate. Here we have computed the RMSE with the ground truth value of test samples and estimated value. I have found out that the RMSE decreases with increasing neighbor i.e, system performs better with increasing neighbor. I have plotted the RMSE of the model by taking the neighbor value from 0 to 100. The plot of RMSE Vs Number of Neighbor is attached here.

The required RMSE for different k values is summarize below-

K value	RMSE
1	12.650396696805034
2	6.369046473792748
3	4.298098563523638
4	3.254650093365229
5	2.599380253105366
10	1.3030515092700221
20	0.9131388926853374
50	0.7112780799947322
100	0.8732971200051043

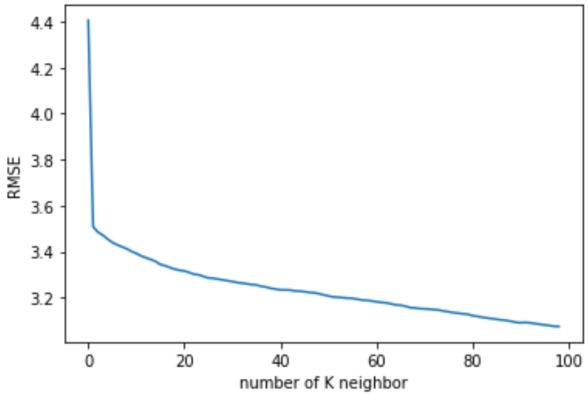
Problem 2.3

The main requirement of this problem is to write 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. The function is attached here,

```
## Code begins here

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

def myLWR(X,Test,k):
    estimate_value_table=[]
    Global_weight_table=[]
    k=k
    Train=X
    test=Test[:,0:2]
    train=Train[:,0:2]
    test_len=test.shape[0]
    train_len=train.shape[0]
    estimate_value_table=[]
    for i in range(test_len):
        distance = []
        for j in range(train_len):
            dist=np.linalg.norm(test[i-1]-train[j-1])
            distance.append([dist,Train[j-1]])
            #Sorted based on distance
        distance.sort(key=lambda x:x[0])
        distance=np.array(distance)

        # Estimate weight
        train_weight=[]
        train_label=[]
        weight=[]
        for m in range(k):
            train_weight.append(distance[m,0:2])
            train_label.append(distance[m,1])
        train_weight=np.mat(train_weight)
        train_label=np.mat(train_label)
        weight= np.matmul(np.matmul(np.linalg.inv(np.matmul(train_weight
   .T, train_weight)), train_weight.T), train_label)
        estimate_value= np.matmul(weight, test[i])
        estimate_value_table.append(estimate_value)

        # finding Accuracy (RMSE)
    sum_rmse=0
    for m in range(test_len):
        sumit=np.square(Test[m-1,2]-estimate_value_table[m])
        sum_rmse=sum_rmse+sumit
    rmse=np.sqrt(sum_rmse/test_len)
    return rmse,estimate_value_table

```

Result evaluation

K Nearest Neighbor (KNN) locally weighted regression is an algorithm which determines it's neighbor based on closest distance and then estimate it's value based on best fit correspond to its neighbor.The weight of the fitting is generated by least square or gradient decent approach to get best value. In this particular algorithm we preferred least square approach to get the weight. Here I have calculated the weight. Then after multiplying with test data, I have got the estimated value. Then I have calculated the RMSE of my method with known vs estimated value. I have plotted the RMSE of the model by taking the neighbor value from 0 to 100. The plot of RMSE Vs Number of Neighbor is attached here.



The required RMSE for different k values is summarize below-

K value	RMSE
1	4.404720332949616
2	3.508749698444397
3	3.4851293600838003
4	3.472133234686119
5	3.4552773504385725
10	3.401319581041007
20	3.3193590947184317
50	3.2126687792388373
100	3.0751857125611988