ECE421 Lab3 Report

1.K-means

1.1 Learning K-means

1)

distance_func

```
# Distance function for K-means
def distance_func(X, mu):
    """ Inputs:
              X: is an NxD matrix (N observations and D dimensions)
              mu: is an KxD matrix (K means and D dimensions)

          Output:
              pair_dist: is the squared pairwise distance matrix (NxK)
    """
    X = tf.expand_dims(X,1) #shape becomes (N,1,D)
    square = tf.square((X-mu))
    pair_dist = tf.reduce_sum(square,2) #change shape to (N,K)
    return pair_dist
```

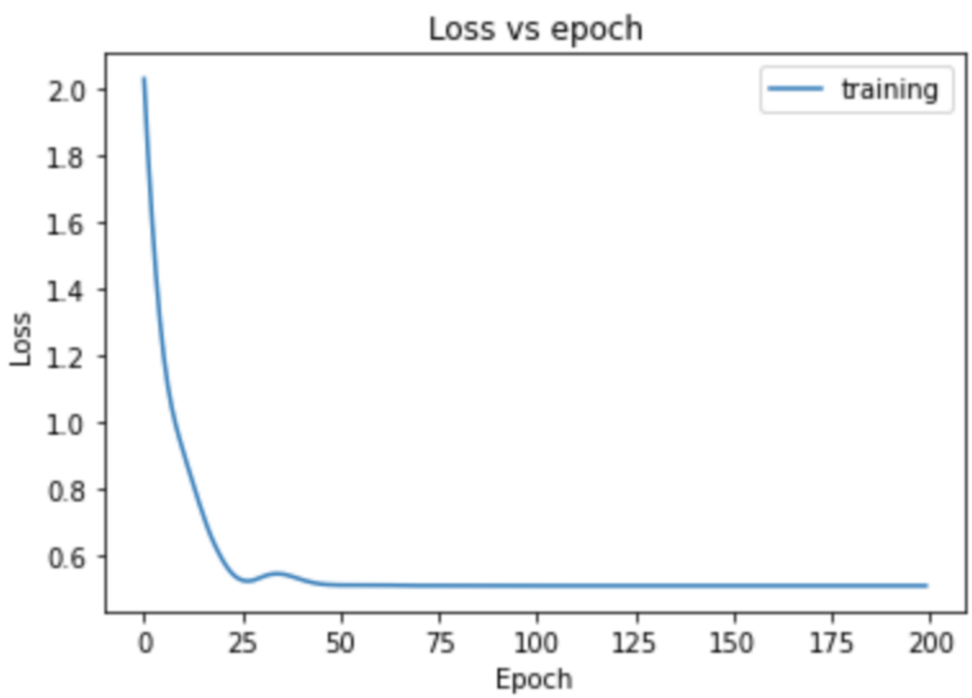
Implement K-means

```
def buildGraph(K,D):
    X = tf.placeholder(tf.float64, shape=(None,D))
    mu = tf.Variable(tf.truncated_normal((K, D), mean=0, stddev=1, dtype=tf.float64), trainable=True)
    loss = tf.reduce_sum(tf.reduce_min(distance_func(X,mu), axis=1))
    optimizer = tf.train.AdamOptimizer(learning_rate=0.1, beta1=0.9, beta2=0.99, epsilon=1e-5).minimize(loss)
    return X,mu,loss,optimizer
```

```
def Train_K_means(dataset,K,epochs,valid_data):
    D = dataset.shape[1]
    X,mu,loss,optimizer = buildGraph(K,D)
    loss_List = []
    best_mu = None
    cluster = None
    valid_loss = None
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for i in range(epochs):
            best_mu,opt = sess.run([mu,optimizer], feed_dict={X: dataset})
            Loss = sess.run(loss, feed_dict={X: dataset})
            loss_List.append(Loss/dataset.shape[0])

        # end of traning find cluster
        # Returns the index with the smallest value across axes of a tensor
        cluster = sess.run(tf.argmin(distance_func(X, best_mu), 1),feed_dict={X:dataset})
        # end of training find validation loss
        valid_loss = sess.run(loss, feed_dict={X: valid_data})
        valid_loss = valid_loss/valid_data.shape[0]
    return loss_List,best_mu,cluster,valid_loss
```

```
def plot_loss(train, valid = None):
    iterations = range(len(train))
    print("train_loss is ", train[-1])
    plt.plot(iterations,train,label = "training")
    if(valid):
        print("valid_loss is ", valid[-1])
        plt.plot(iterations,valid,label = "validation")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend()
    plt.title('Loss vs epoch')
    plt.show()
    return

def Q1():
    data = load_dataset(True)
    loss_List,best_mu,cluster,valid_loss = Train_K_means(data,3,200,data)
    plot_loss(loss_List)
    return
```

```
train_loss is  0.511094537291807
```

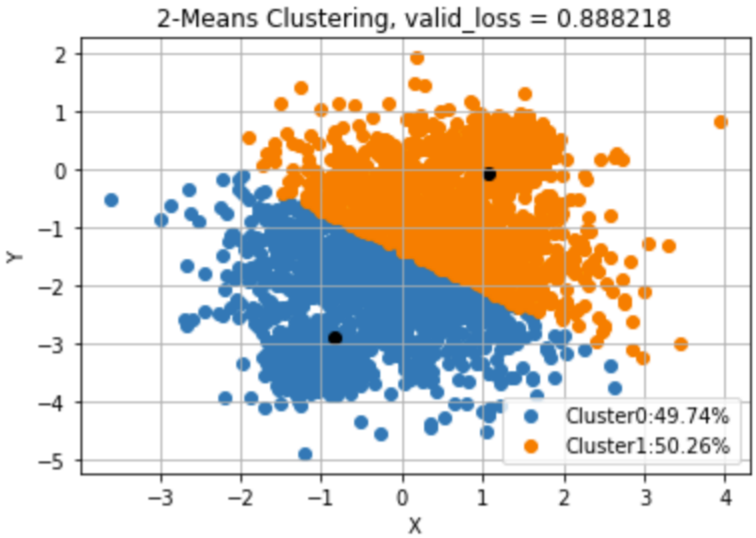


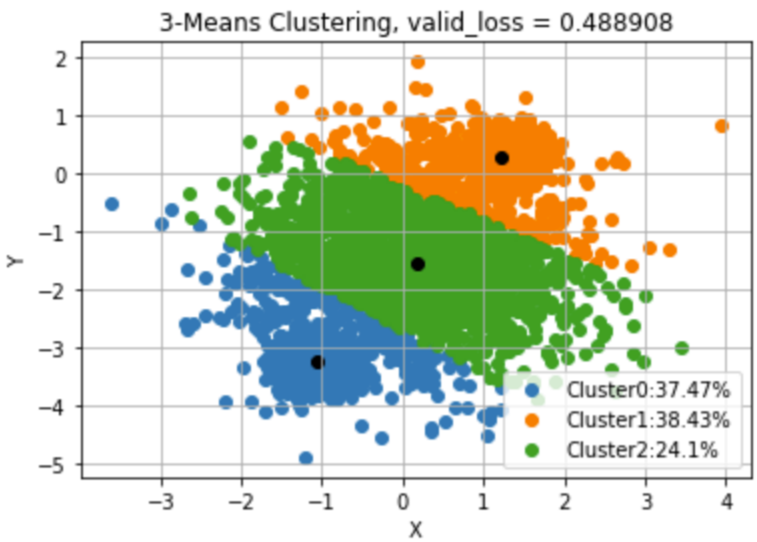
2)

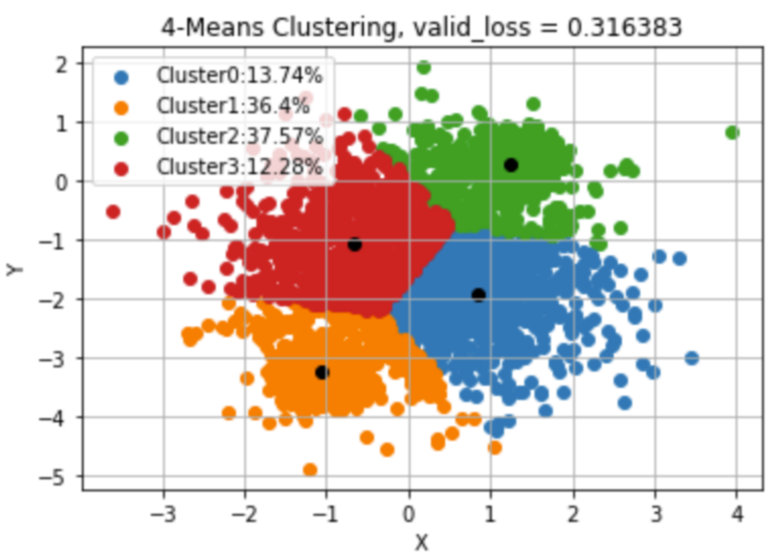
```
def  plot_cluster(K,dataset,cluster,mu,valid_loss):
    legend  =  []
    for  i  in  range(K):
        class_i=[]
        for  j  in  range(len(cluster)):
            if  (cluster[j]==i):
                class_i.append(dataset[j,:])
        class_i  =  np.array(class_i)
        plt.scatter(class_i[:,   0],   class_i[:,   1],   cmap='Pastel')
        #calculate  number  of  data  belongs  to  cluster
        percentage  =  str(round(100*np.sum(i==cluster)/len(cluster),   2))
        legend.append("Cluster"+str(i)+":"+percentage+"%")
    plt.legend(legend)
    plt.scatter(mu[:,   0],   mu[:,   1],   c='black')
    plt.title("%d-Means  Clustering,   valid_loss  =  %f"  %(K,valid_loss))
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.grid()
    plt.show()
    return
```

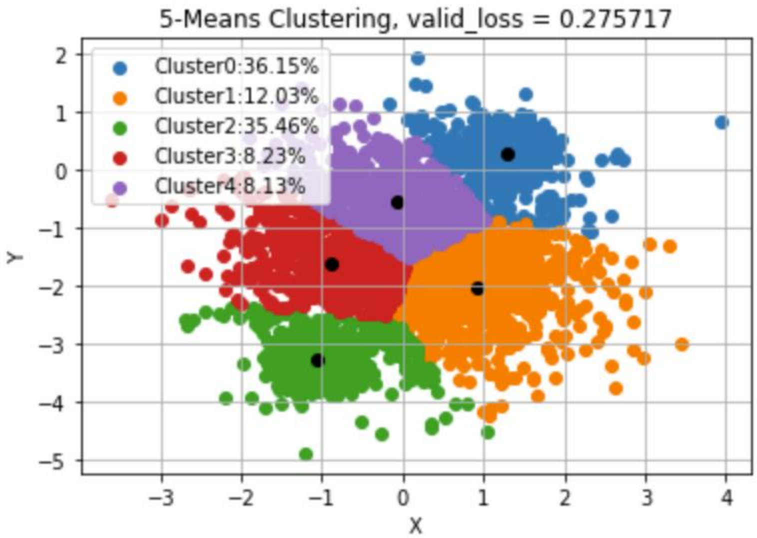
```
def  Q2():
    K_list  =  [1,2,3,4,5]
    for  K  in  K_list:
        data,   val_data  =  load_dataset(True,True)
        loss_List,best_mu,cluster,valid_loss  =  Train_K_means(data,K,100,val_data)
        plot_cluster(K,data,cluster,best_mu,valid_loss)
    return
```

Plot


2-Means Clustering, valid_loss = 0.888218
Cluster0:49.74%
Cluster1:50.26%
X
Y


3-Means Clustering, valid_loss = 0.488908
Cluster0:37.47%
Cluster1:38.43%
Cluster2:24.1%
X
Y


4-Means Clustering, valid_loss = 0.316383
Cluster0:13.74%
Cluster1:36.4%
Cluster2:37.57%
Cluster3:12.28%
X
Y



From the figures above,

Number of Cluster - K	Validation Loss
1	3.861
2	0.888
3	0.488
4	0.316
5	0.275

The minimum validation loss achieved here is when K=5, validation loss = 0.275

The validation loss decreases as K increase.

However, it doesn't make sense to divide the dataset into too many clusters.

The validation loss decreases drastically as K increase from 1 to 3, the validation loss drops 3.4.

The validation loss decreases much slower when increase K from 3 to 5, the validation loss drops 2.1.

Thus, I think K =3 is the optimal number of clusters.

2.Mixtures of Gaussians

2.1 Gaussian cluster mode

1)

$$N(x, \mu k, \delta k) = \frac{1}{\delta k\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu k}{\delta k}\right)^2\right)$$

$$\log\big(N(x, \mu k, \delta k)\big) = -\log\left(\delta k\sqrt{2\pi}\right) - \frac{1}{2}\left(\frac{x-\mu k}{\delta k}\right)^2$$

$$= -\frac{1}{2}\log(2\pi\,\delta k^2) - \frac{1}{2}\left(\frac{x-\mu k}{\delta k}\right)^2$$

```
def log_gauss_pdf(X, mu, sigma):
    """ Inputs:
            X: N X D
            mu: K X D
            sigma: K X 1

        Outputs:
            log Gaussian PDF (N X K)
    """
    part1 = -(0.5) * tf.log(2 * np.pi * tf.transpose(sigma)**2)
    pair_dist = distance_func(X, mu)
    part2 = -(0.5) * tf.square(pair_dist) / (tf.transpose(sigma)**2)
    return part1 + part2
```

2)

$$P(z|x) = \frac{P(x|z)P(z)}{\sum_i^k P(x|i)P(i)}$$

$$logP(z|x) = \log\left(P(x|z)P(z)\right) - \log\left(\sum_i^k P(x|i)P(i)\right)$$

$$= \log P(x|z) + P(z) - \log\left(\sum_i^k P(x|i)P(i)\right)$$

$$= \log P(x|z) + P(z) - \log\left(\sum_i^k \exp\big(logP(x|i) + logP(i)\big)\right)$$

```
def log_posterior(log_PDF, log_pi):
    """ Inputs:
            log_PDF: log Gaussian PDF N X K
            log_pi: K X 1

        Outputs
            log_post: N X K
    """
    log_numerator = log_PDF + tf.squeeze(log_pi)
    log_denominator = reduce_logsumexp(log_numerator, keep_dims=True)
    answer = log_numerator - log_denominator
    return answer
```

The reason to use `reduce_logsumexp` since we need to compute

$$\log\left(\sum_{i}^{k}\exp\left(logP(x|i)+logP(i)\right)\right)$$

while `tf. reduce_sum` only compute the sum

2.2 Learning the MoG

1)

```
def buildGraph(K,D):
    #define variables
    X = tf.placeholder(tf.float32, shape=(None,D))
    mu = tf.Variable(tf.random_normal([K, D], stddev = 1))
    # theta is unconstrained parameter, sigma = exp(phi) with [0 - inf]
    theta = tf.Variable(tf.random_normal([K, 1], stddev = 1))
    sigma = tf.exp(theta)

    # phi is unconstrained parameter, acheive constraint for pi
    phi = tf.Variable(tf.random_normal([K, 1], stddev = 1))
    log_pi = logsoftmax(phi)

    log_pdf = log_gauss_pdf(X,mu,sigma)

    #defien loss & optimizer
    loss= - tf.reduce_sum(reduce_logsumexp(log_pdf + tf.squeeze(log_pi), keep_dims=True))
    optimizer = tf.train.AdamOptimizer(learning_rate=0.1, beta1=0.9, beta2=0.99, epsilon=1e-5).minimize(loss)
    return X,mu,sigma,optimizer,loss,log_pdf,log_pi
```

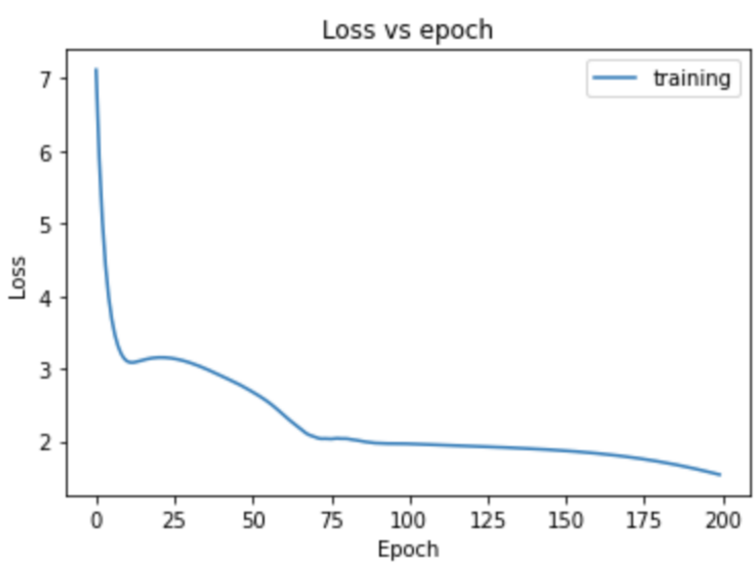
```
def  Train_GMM_means(dataset,K,epochs,valid_data):
    D  =  dataset.shape[1]
    X,mu,sigma,optimizer,loss,log_pdf,log_pi  =  buildGraph(K,D)
    loss_List  =  []
    best_mu  =  None
    best_sigma  =  None
    cluster  =  None
    valid_loss  =  None
    with  tf.Session()  as  sess:
            sess.run(tf.global_variables_initializer())
            for  i  in  range(epochs):
                    best_mu,best_sigma,opt  =  sess.run([mu,sigma,optimizer],  feed_dict={X:  dataset})
                    Loss  =  sess.run(loss,  feed_dict={X:  dataset})
                    loss_List.append(Loss/dataset.shape[0])

            #  end  of  traning  find  cluster
            #  Returns  the  index  with  the  smallest  value  across  axes  of  a  tensor
            cluster  =  sess.run(tf.argmax(log_posterior(log_pdf,log_pi),1),feed_dict={X:dataset})
            #  end  of  training  find  validation  loss
            valid_loss  =  sess.run(loss,  feed_dict={X:  valid_data})
            valid_loss  =  valid_loss/valid_data.shape[0]
```

```
return  loss_List,best_mu,best_sigma,cluster,valid_loss
```

```
def  Q1():
    data  =  load_dataset(True)
    loss_List,best_mu,best_sigma,cluster,valid_loss  =  Train_GMM_means(data,3,200,data)
    plot_loss(loss_List)
    print("best_mu:",best_mu)
    print("best_sigma",best_sigma)
    return
```

Plot Loss:

```
train_loss is  1.53743203125
```

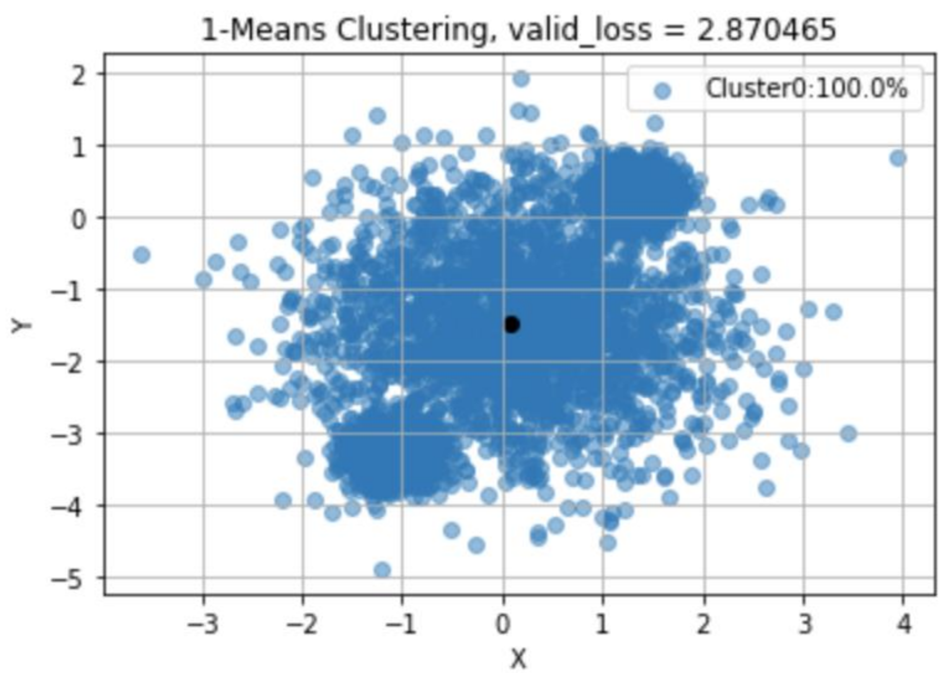


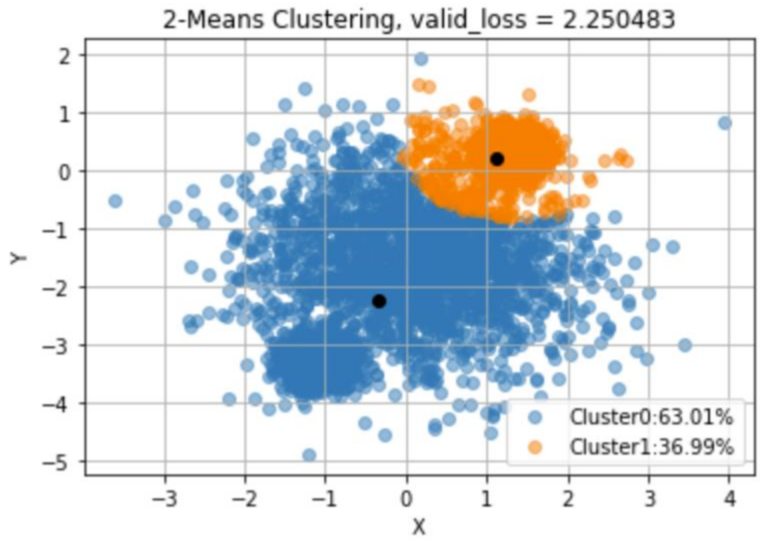
Model Parameter:

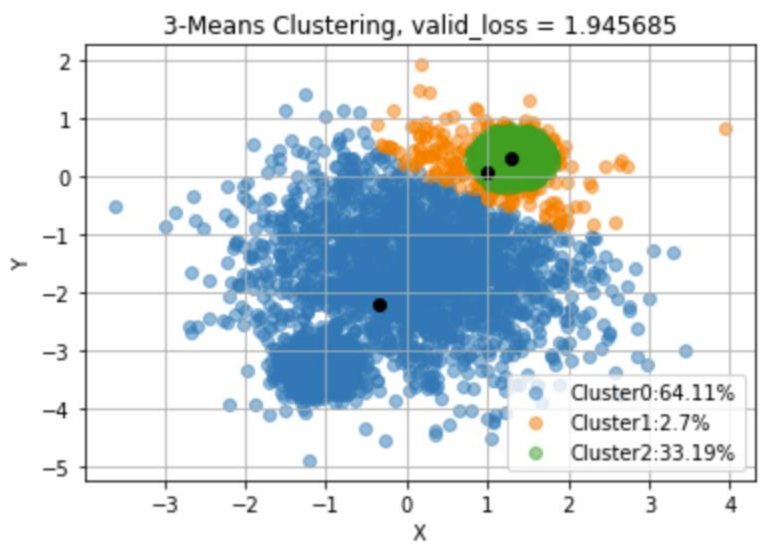
Cluster K	Log pi	mu	Sigma
1	-1.0067024	[0.05538136 -1.6027924]	2.9715424
2	-1.1738701	[-1.1191827 -3.316661]	0.07645379
3	-1.1226696	[1.3025047 0.31507367]	0.09755903

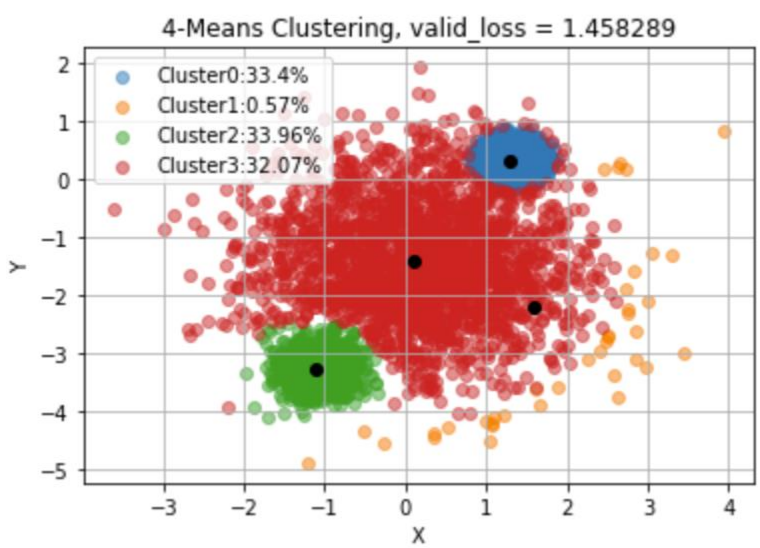
2)

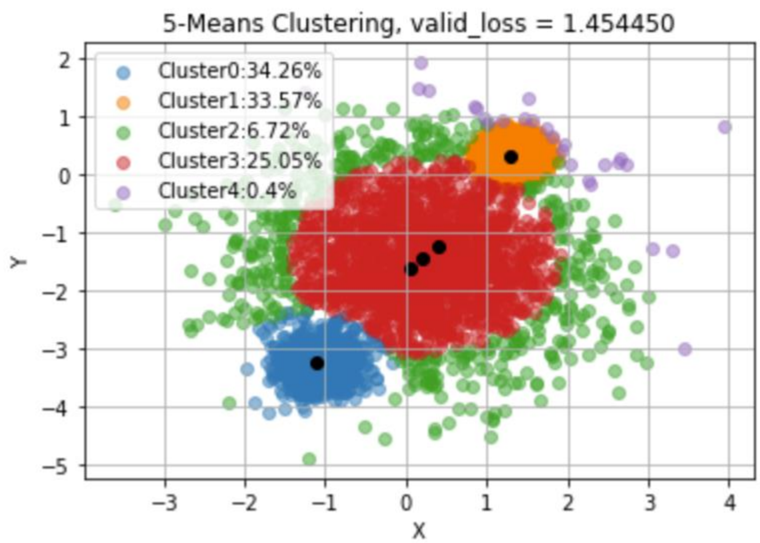
```
def Q2():
    K_list = [1,2,3,4,5]
    for K in K_list:
        data, val_data = load_dataset(True,True)
        loss_List,best_mu,best_sigma,best_pi,cluster,valid_loss = Train_GMM_means(data,K,100,val_data)
        plot_cluster(K,data,cluster,best_mu,valid_loss)
    return
```




2-Means Clustering, valid_loss = 2.250483
Cluster0:63.01%
Cluster1:36.99%
X
Y


3-Means Clustering, valid_loss = 1.945685
Cluster0:64.11%
Cluster1:2.7%
Cluster2:33.19%
X
Y


4-Means Clustering, valid_loss = 1.458289
Cluster0:33.4%
Cluster1:0.57%
Cluster2:33.96%
Cluster3:32.07%
X
Y



From the figures above,

Number of Cluster - K	Validation Loss
1	2.87
2	2.25
3	1.945
4	1.458
5	1.454

The minimum validation loss achieved here is when K=5, validation loss = 1.454

The validation loss decreases as K increase.

However, it doesn’t make sense to divide the dataset into too many clusters.

The validation loss decreases drastically as K increase from 1 to 3, the validation loss drops 0.93.

The validation loss decreases much slower when increase K from 3 to 5, the validation loss drops 0.39.

Thus, I think K =3 is the optimal number of clusters.

3) Compare K-means and MoG

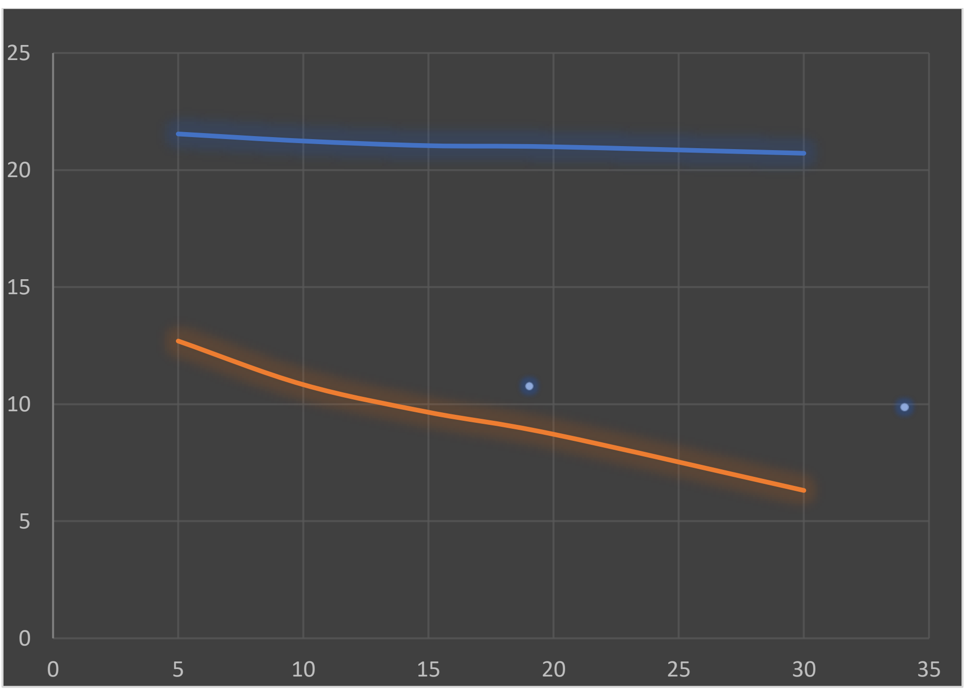
K	k-means Validation Loss	MoG Validation Loss
5	21.54334444037717	12.697508813381338
10	21.24050911505024	10.834202951545155
15	21.042035154237134	9.654315040879087
20	20.992872165406673	8.715772553817882
30	20.71745446427862	6.316014218609361

The minimum validation loss for both k-means and MoG achieved here is when K=30.

k-means validation loss = 20.717   MoG validation loss = 6.316

The validation loss for both k-means and MoG decreases as K increase.

However, increase in K post more effect on MoG algorithm, since its validation loss decrease more rapidly comparing to the validation loss for k-means as K increase.



From the graph above , we can see that both k-means and Mog validation loss decrease with a steady rate as K increase thus, I think K =30 is the optimal number of clusters.