ECE421 – Introduction to Machine Learning

Assignment 3

Unsupervised Learning and Probabilistic Models

Hard Copy Due: Friday, March 27 @ 3:00 PM, at BA3128
Code Submission Due: Friday, March 27 @ 5:00 PM, on Quercus

General Notes:

- Attach this cover page to your hard copy submission
- Please post assignment related questions on Piazza.

Please check section to which you would like the assignment returned.

Tutorial Sections:

☐ Tutorial 1: Thursdays 3-5pm (SF2202)
☑ Tutorial 2: Thursdays 3-5pm (GB304)
☐ Tutorial 3: Tuesdays 10-12 (SF2202)
☐ Tutorial 4: Fridays 9-11 (BA1230)

Group Members	
Name	Student ID
Alexander Mark	1003974185
Sharon Lin	1004099254
Kyu Bum Kim	1003969100

Contribution: Every partner contributed an equal amount of work (33/33/33)%

Part 1: K-Means

1.1 Learning K-Means

1.

DistanceFunc() formula: $\sqrt{\Sigma_{i=1}^{D}(x_n^i - \mu_k^i)^2} \quad \forall\ n, k$

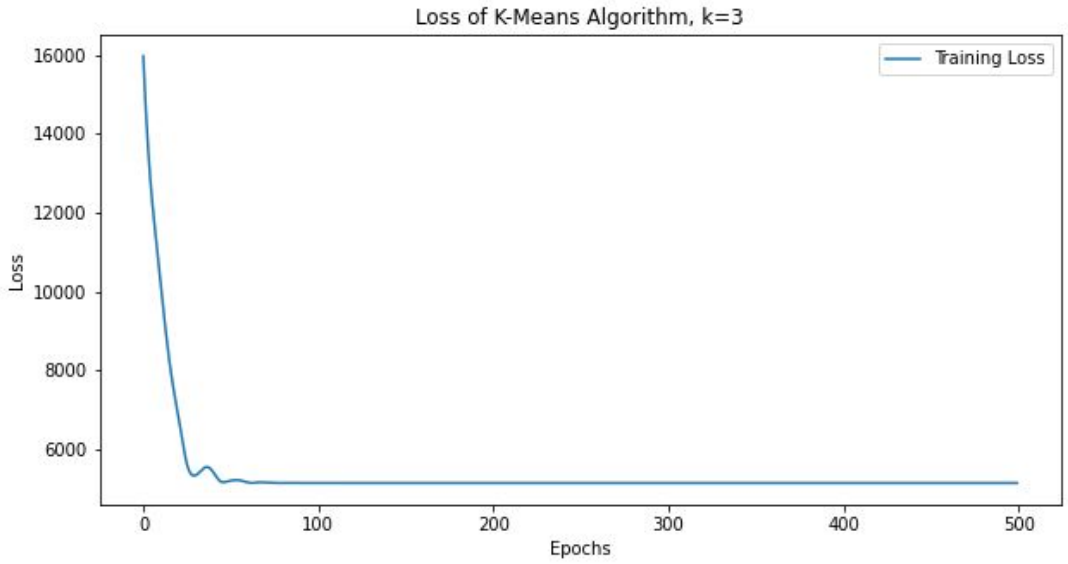
$x = N \times D$ data points

$\mu = K \times D$ cluster centers

$output = N \times K$ euclidean distance between each point and cluster center

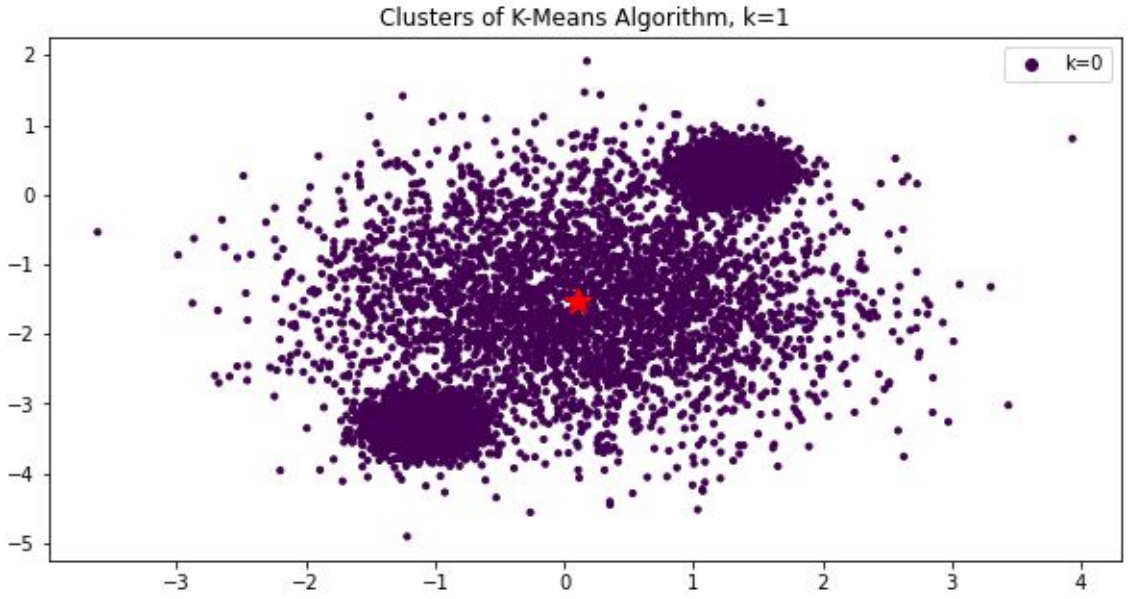
```
def distanceFunc(X, MU):
    # calculate differences between all pairs of points
    diffs = tf.expand_dims(X, -1) - tf.transpose(MU)
    dist = tf.norm(diffs, axis=1)
    return dist
```

Using K-Means Algorithm for K=3

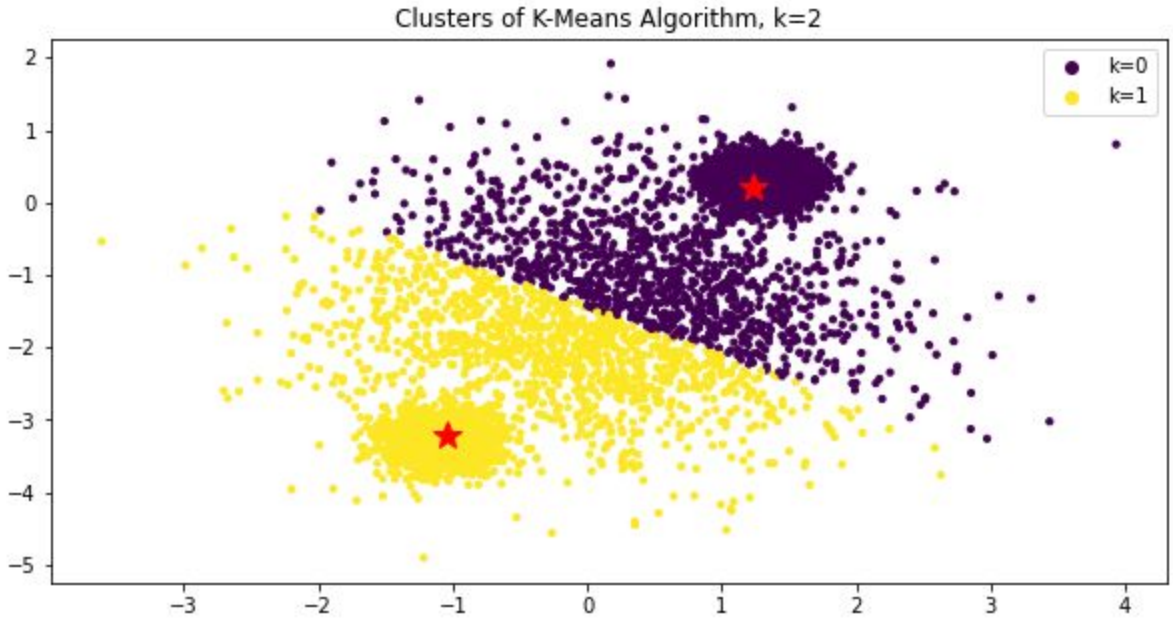


The final loss = 5146.15

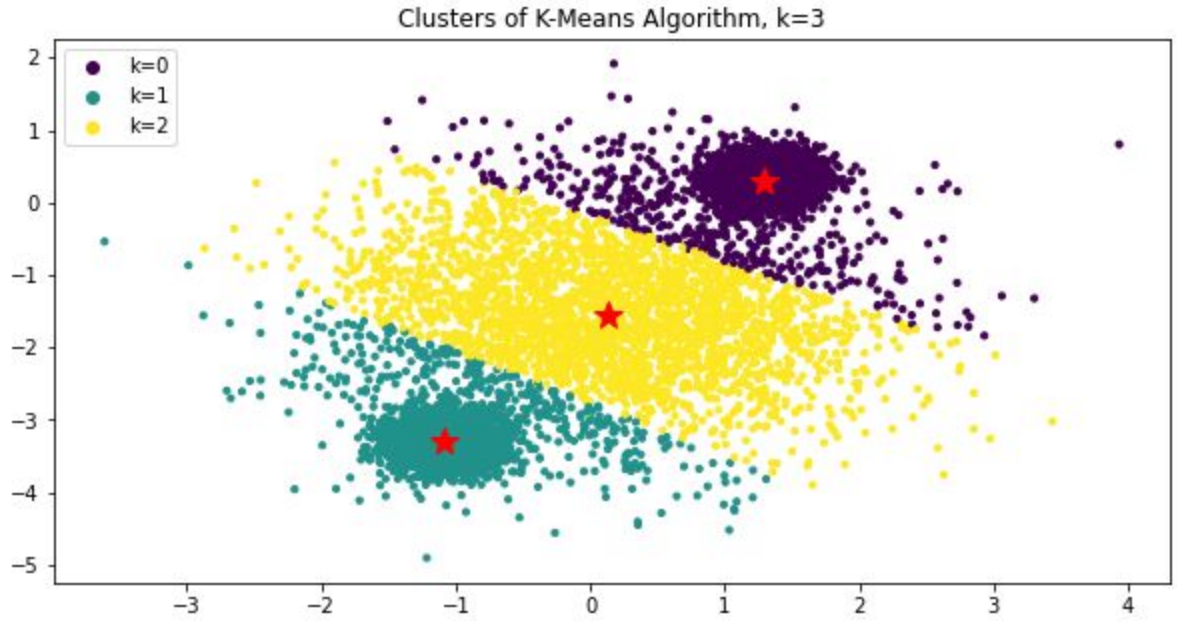
2. Using K-Means Algorithm for K=1 (the stars indicate the cluster center)



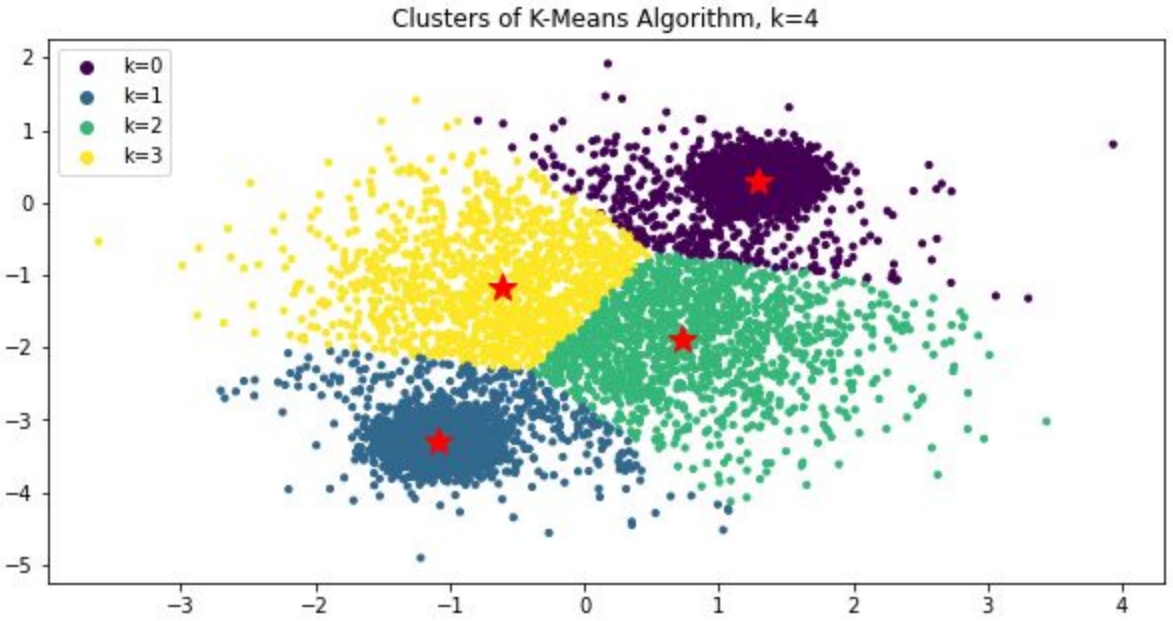
Using K-Means Algorithm for K=2 (the stars indicate the cluster center)



Using K-Means Algorithm for K=3 (the stars indicate the cluster center)



Using K-Means Algorithm for K=4 (the stars indicate the cluster center)



Using K-Means Algorithm for K=5 (the stars indicate the cluster center)

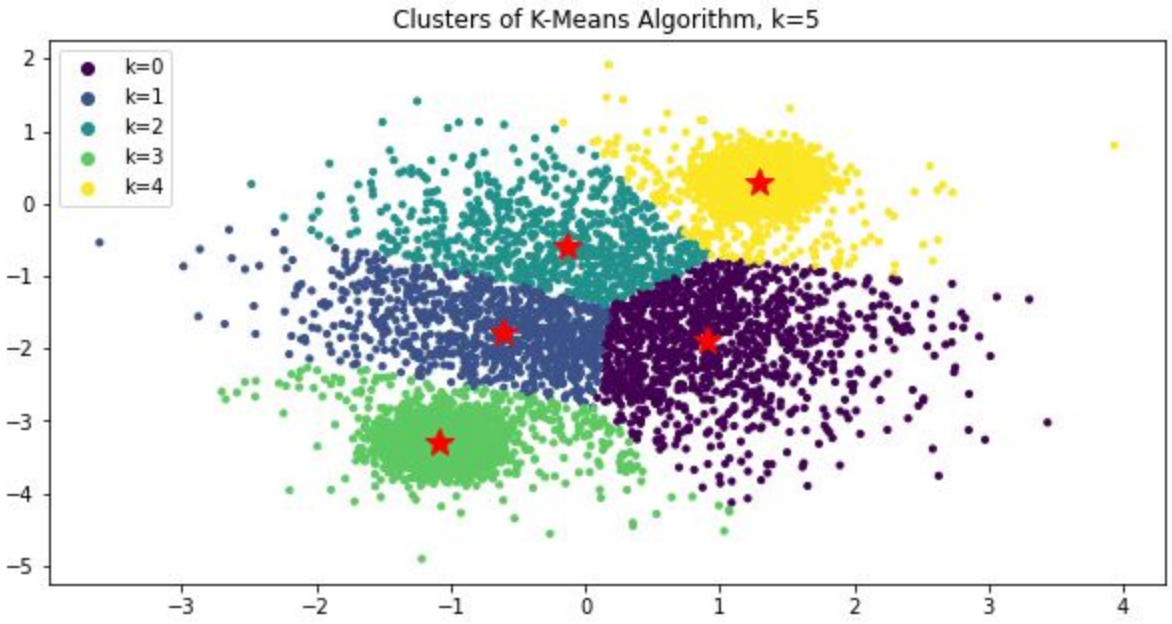


Table 1: Percentage of Points Belonging to Each Cluster

K	**Cluster 1**	**Cluster 2**	**Cluster 3**	**Cluster 4**	**Cluster 5**
1	100%	0	0	0	0
2	50.38%	49.62%	0	0	0
3	37.89%	37.86%	24.25%	0	0
4	37.2%	36.92%	13.72%	12.16%	0
5	11.18%	9.14%	7.78%	36.21%	35.69%

The graph with 3 clusters is probably best because it has the most equal percentage of points assigned to each cluster (38/38/24). At K=4 or K=5, the same 2 clusters from K=3 remain intact while the third one just gets broken into 2 or 3 clusters. This is not ideal.

3.

Table 2: Final Validation Loss for Varying K

K	**Validation Loss**	**|∆ Validation Loss|**
1	6252.75	-
2	2340.67	3912.08
3	1681.82	658.85
4	1452.23	199.59
5	1359.26	92.97

As K increases, the validation loss should continue decreasing in an exponentially decaying form. We can compare the magnitudes in validation loss difference to see where we should stop increasing K. If the difference is insubstantial, then we should stop there. One can observe that after K=3, the difference is marginal (only reduction of 200 in loss) so K=3 should be ideal.

Part 2: Mixtures of Gaussians (MoG)

2.1 The Gaussian Cluster Mode

1.

Multivariate Normal Distribution:

$$\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Multivariate Normal Log Distribution:

$$ln\left(\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)\right)$$

Multivariate Normal Log Distribution Simplification:

$$ln\left(\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}}\right) + ln\left(exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)\right)$$

$$ln\left((2\pi)^{-\frac{d}{2}}|\Sigma|^{-\frac{1}{2}}\right) - \frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)$$

$$-\frac{1}{2}ln\left((2\pi)^d|\Sigma|\right) - \frac{(x-\mu)^2}{2\Sigma}$$

$x = N \times D$ data points

$\mu = K \times D$ cluster centers

$\sigma^2 = K \times 1$ variances (the diagonal on the Σ matrix; all other values are 0)

$output = N \times K$ log Gaussian PDF

Normal Log PDF Python Implementation:

```
def log_GaussPDF(X, mu, sigma):
    D = tf.cast(tf.rank(X), tf.float32)
    x_mu2 = distanceFuncGMM(X, mu)
    sigma = tf.squeeze(sigma)

    term1 = -1 * 0.5 * tf.log(((2 * np.pi)**D) * sigma)
    term2 = -1 * x_mu2 / (2 * sigma)

    return term1 + term2
```

Where x_mu2 is using a modified distance function which calculates:

$$\Sigma_{i=1}^{D}(x_n^i - \mu_k^i)^2 \quad \forall\ n, k$$

Updated Distance Function Python Implementation:

```
def distanceFuncGMM(X, MU):
    diffs = tf.expand_dims(X, -1) - tf.transpose(MU)
    x_mu2 = tf.reduce_sum(tf.square(diffs), axis=1)
    return x_mu2
```

2.

In order to calculate the probabilities, we must use:

$$log\,(P(z|X)) = log\left(\frac{P(X|z)P(z)}{P(X)}\right) = log(P(X|z)) + log(P(z)) - log(P(x)) \tag{1}$$

The above is found using Bayes' theorem

We know that this term, $log(P(X|z))$, is equivalent to the normal log multivariate distribution. We also know that $log(P(z)) = log(\pi)$ is equivalent to the log probability of cluster variables. Because we want the probability of each cluster variable corresponding to a given point, the joint log probability is:

$$log(P(X|z)) + log(P(z))$$

To find the last term:

$$log(P(x)) = log\left(\sum_{k=1}^{K} \pi_k N(x_i; \mu_k, \sigma_k^2)\right)$$

Which can be simplified to:

$$log\left(\sum_{k=1}^{K} (log(P(X|z)) + log(P(z)))\right) \tag{2}$$

Which can be solved using *reduce_logsumexp*, thus, this function is required. Also, using the reduce_logsumexp function instead of reduce_sum will avoid overflow/underflow issues resulting from doing calculations with numbers of small probabilities. This results in the code below which uses the form from (1) using the new simplified values found in (2):

```
def log_posterior(log_PDF, log_pi):
    log_pi = tf.squeeze(log_pi)
    return log_PDF + log_pi - reduce_logsumexp(log_PDF + log_pi)
```

2.2 Learning the MoG

1.

In order to obtain the loss, we must calculate the negative log likelihood:

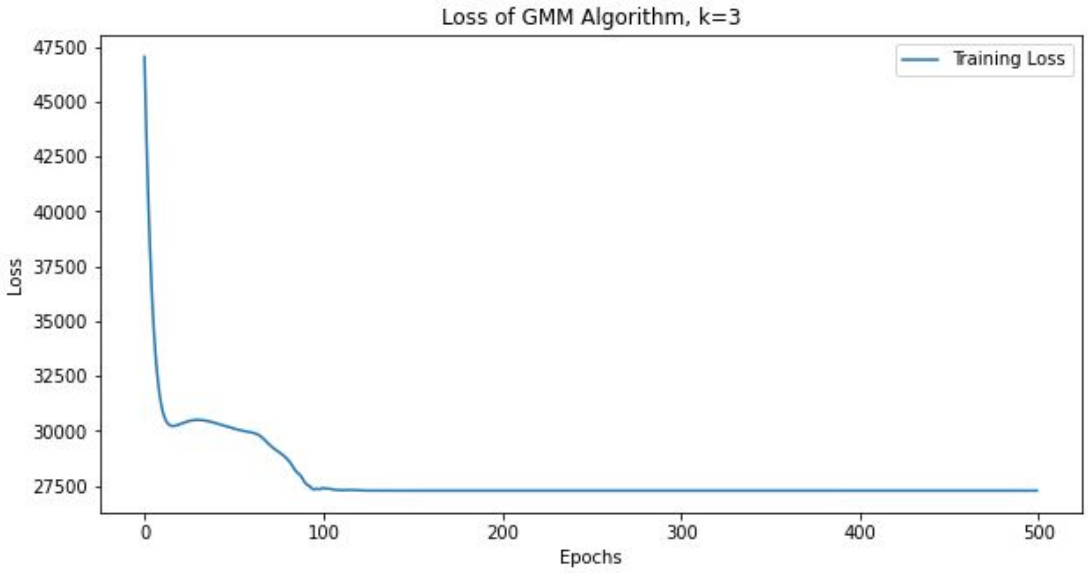
$$-log(P(X)) = -log\prod_{n=1}^{N} P(x_n) \quad (3)$$

We obtained (3) through equation (2) and the lab handout. Therefore, this can be simplified.

$$-log(P(x_1)P(x_2)...P(x_N))$$
$$-log(P(x_1)) - log(P(x_2)) - ... - log(P(x_N))$$
$$-\sum_{n=1}^{N} log(P(x_n))$$

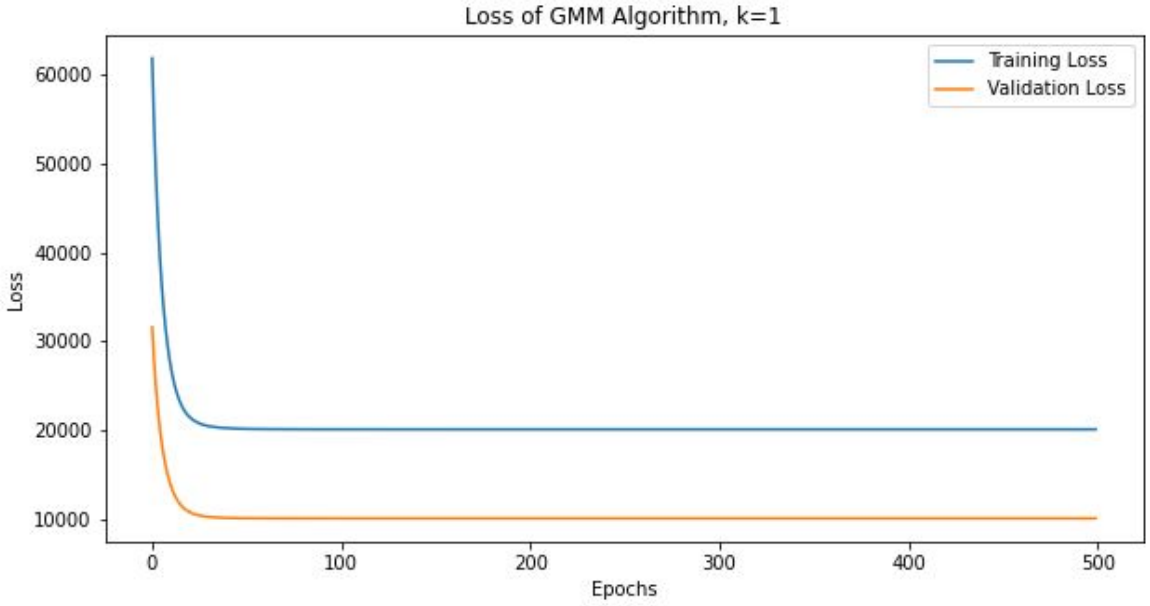
Which requires the reduce_logsumexp function. Within the GMM graph function, we implemented a loss function:

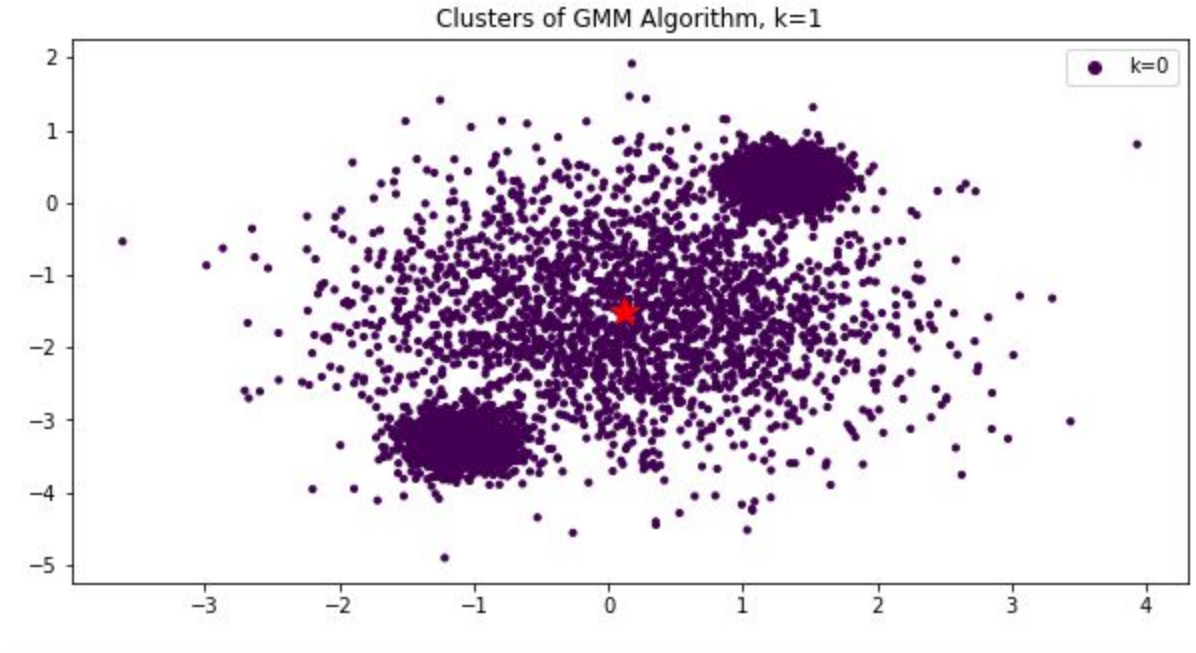
```
def GMMGraph(K, D):
    stddev = 1
    ...
    data = tf.placeholder(tf.float32, shape=(None, D), name="trainData")
    Mu = tf.Variable(tf.random.normal(shape=[K, 1], stddev=stddev),
        name="mu")

    phi = tf.Variable(tf.random.normal(shape=[K, 1], stddev=stddev),
        name="Phi")
    phi_exp = tf.math.exp(phi)
    prob = logsoftmax(phi_exp)

    ln_gauss = log_GaussPDF(data, Mu, phi_exp)
    multiplication = reduce_logsumexp(ln_gauss + tf.squeeze(prob),
        reduction_indices=1)
    loss = -1 * tf.reduce_sum(multiplication, reduction_indices=0)
    ...
```


Loss of GMM Algorithm, k=3
Training Loss
47500
45000
42500
40000
37500
35000
32500
30000
27500
Loss
0
100
200
300
400
500
Epochs

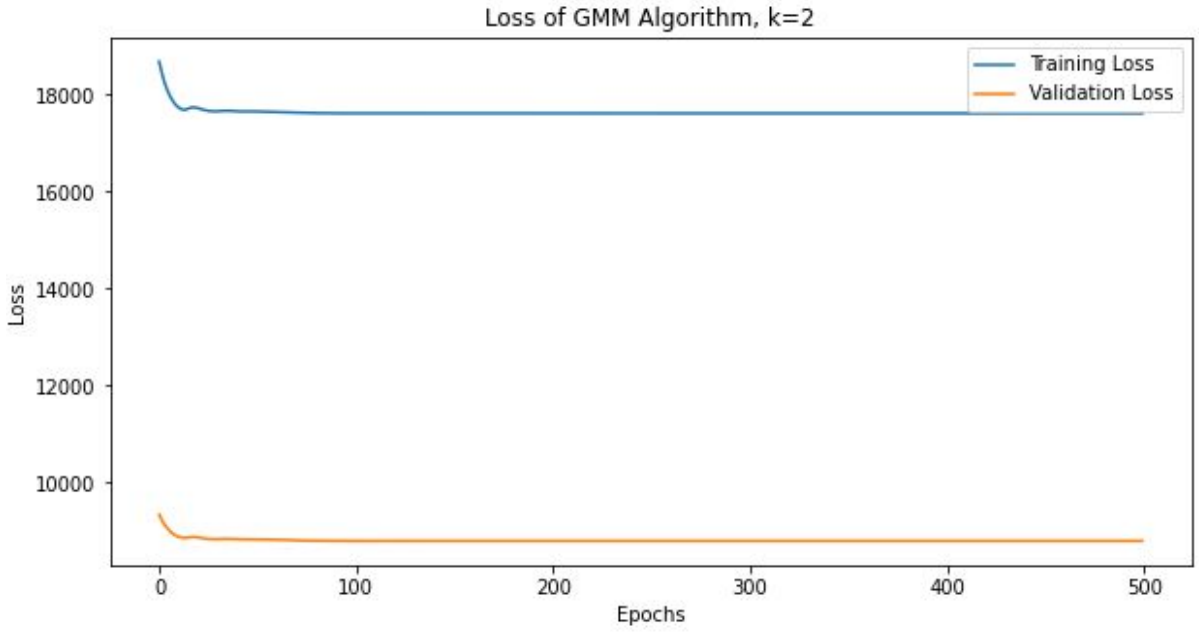
2.

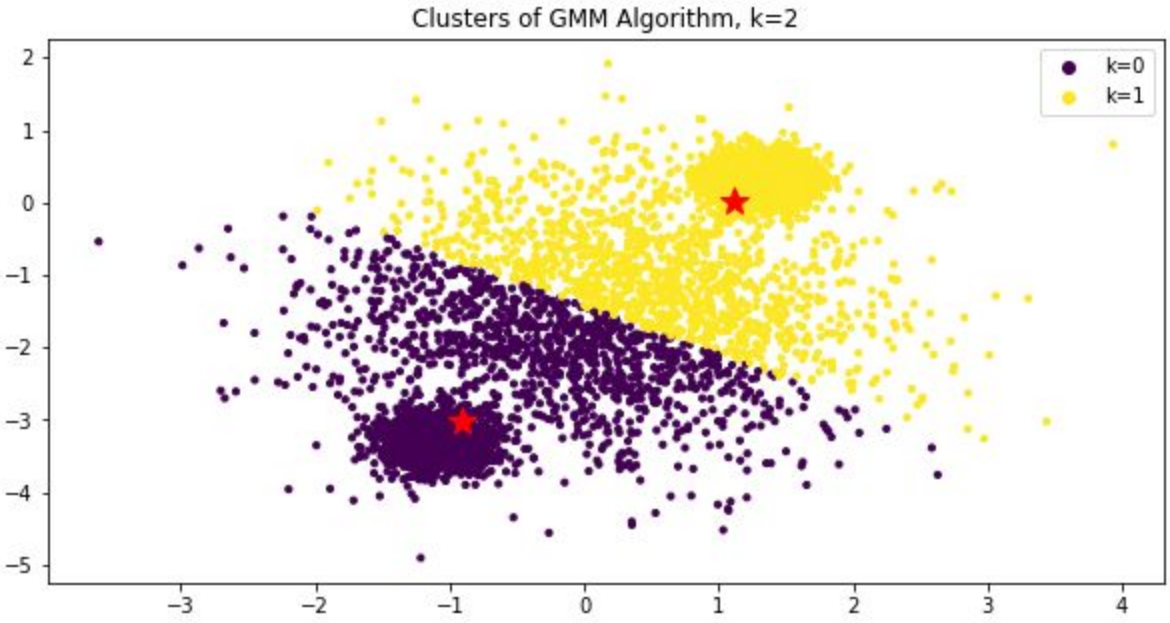
GMM for K=1


Loss of GMM Algorithm, k=1
Training Loss
Validation Loss
60000
50000
40000
30000
20000
10000
Loss
0
100
200
300
400
500
Epochs

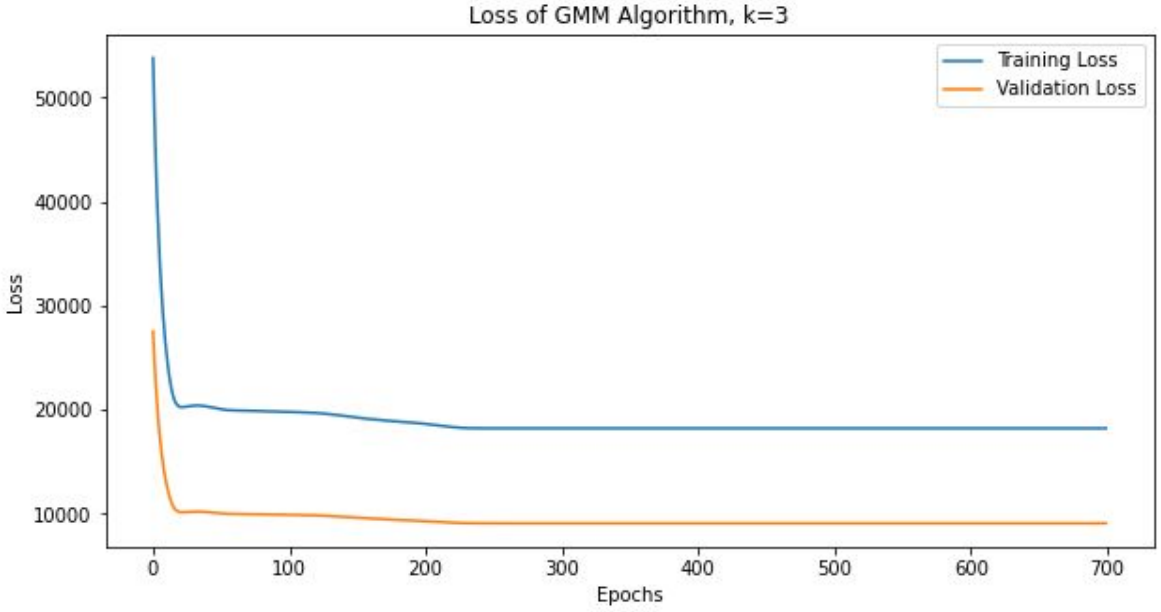

Clusters of GMM Algorithm, k=1
k=0
2
1
0
−1
−2
−3
−4
−5
−3
−2
−1
0
1
2
3
4

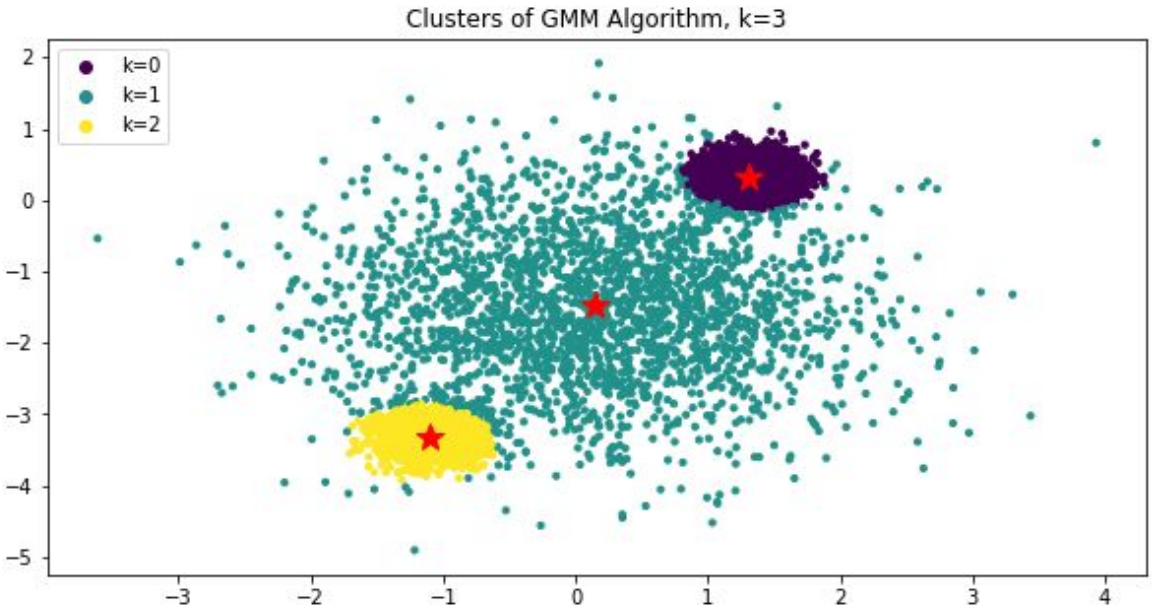
GMM for K=2


Loss of GMM Algorithm, k=2
Training Loss
Validation Loss
18000
16000
14000
12000
10000
Loss
0
100
200
300
400
500
Epochs

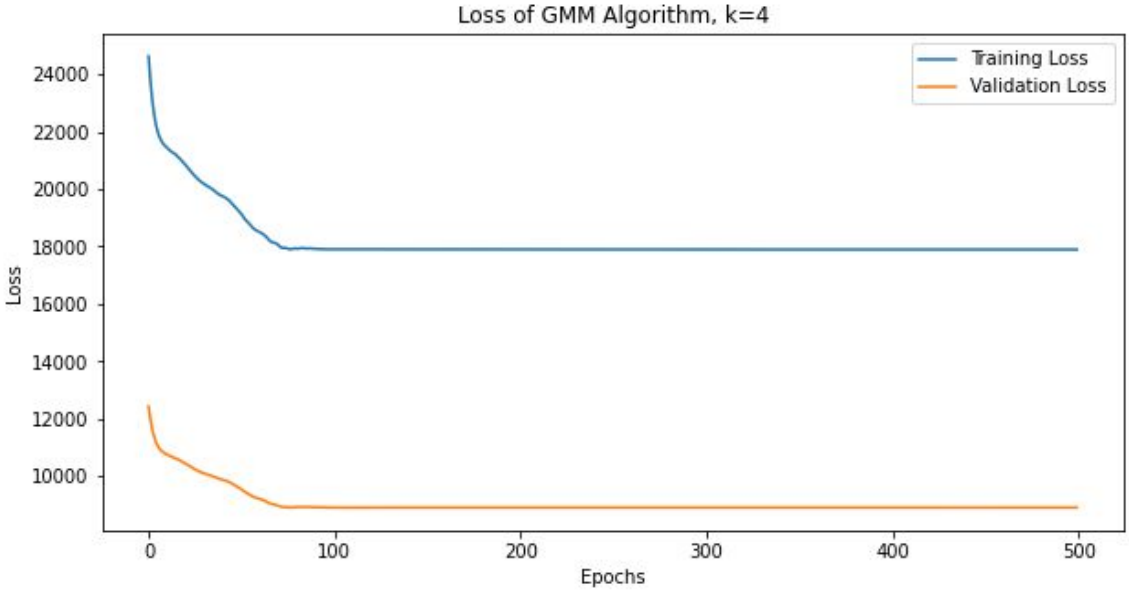

Clusters of GMM Algorithm, k=2
k=0
k=1
2
1
0
−1
−2
−3
−4
−5
−3
−2
−1
0
1
2
3
4

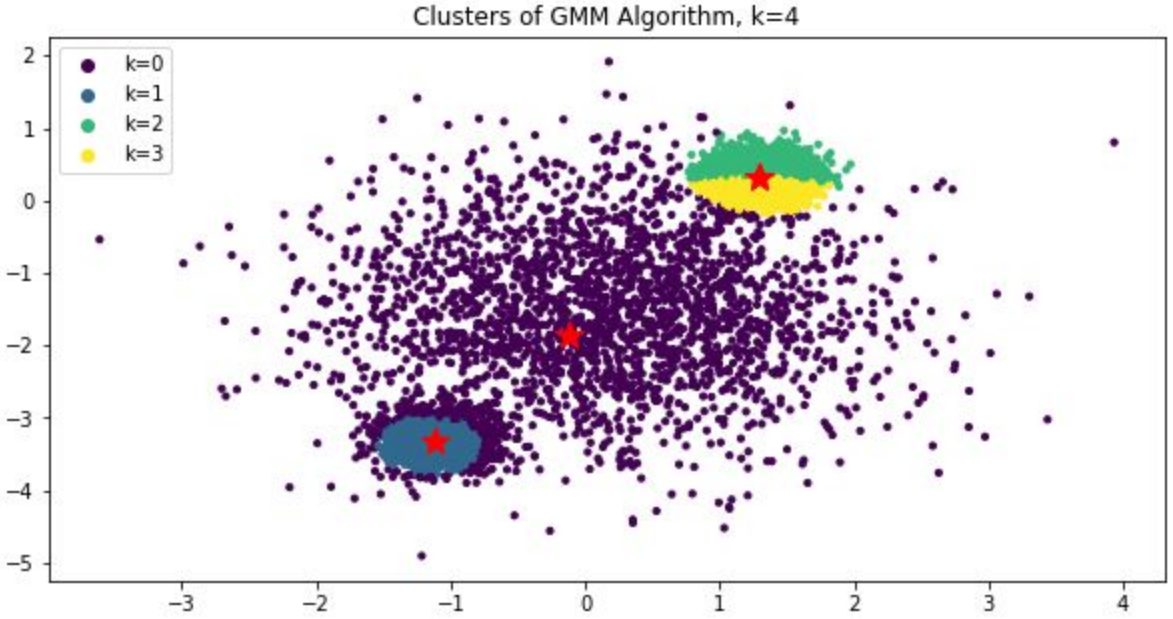
GMM for K=3


Loss of GMM Algorithm, k=3
Training Loss
Validation Loss
50000
40000
30000
20000
10000
Loss
0
100
200
300
400
500
600
700
Epochs

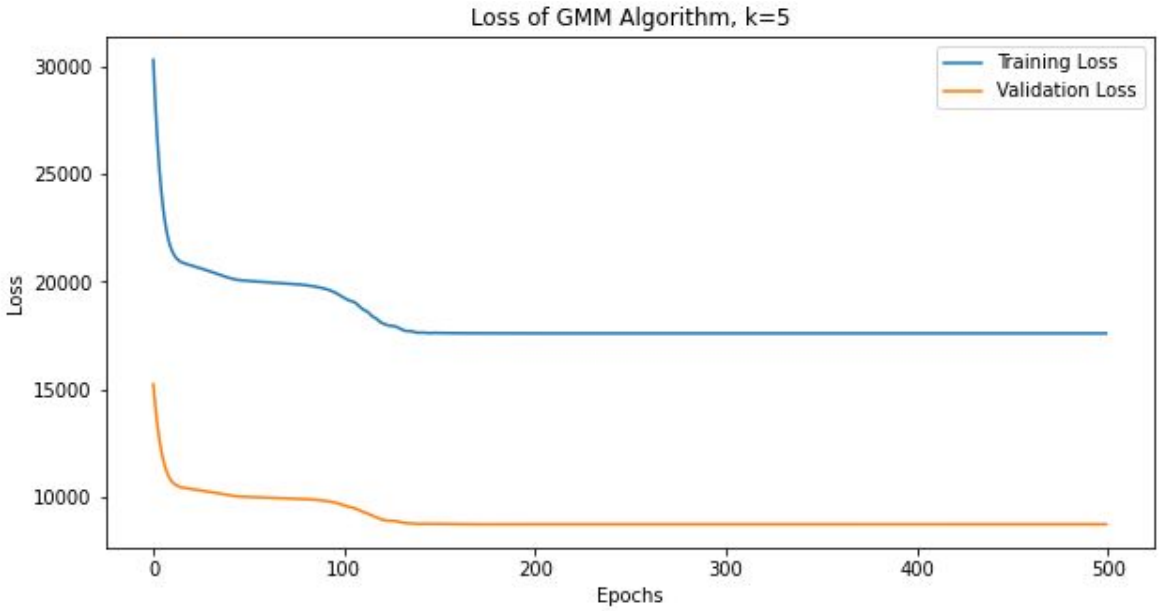

Clusters of GMM Algorithm, k=3
k=0
k=1
k=2
2
1
0
−1
−2
−3
−4
−5
−3
−2
−1
0
1
2
3
4

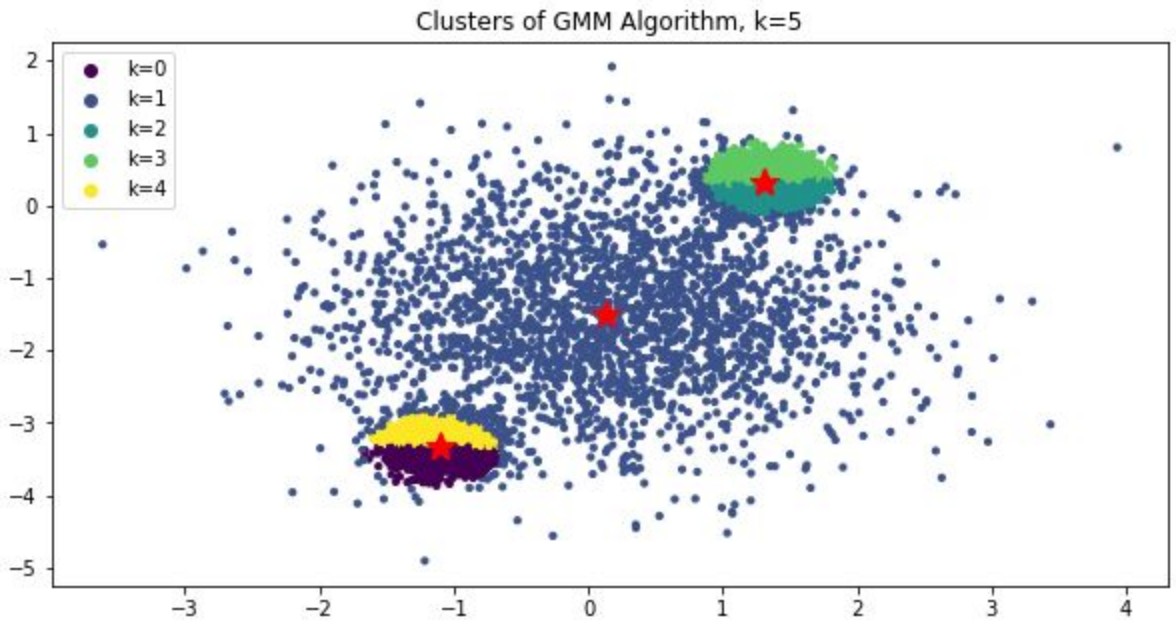
GMM for K=4


Loss of GMM Algorithm, k=4
Training Loss
Validation Loss
Loss
24000
22000
20000
18000
16000
14000
12000
10000
0
100
200
300
400
500
Epochs


Clusters of GMM Algorithm, k=4
k=0
k=1
k=2
k=3
2
1
0
−1
−2
−3
−4
−5
−3
−2
−1
0
1
2
3
4

GMM for K=5


Loss of GMM Algorithm, k=5
Training Loss
Validation Loss
30000
25000
20000
15000
10000
Loss
0
100
200
300
400
500
Epochs



K=3 is the best number of clusters as the validation loss did not decrease substantially beyond K=3. At K=4 and K=5, the validation loss only decreased by 175 and 145 respectively which makes a marginal difference. Therefore, K=3 is best.

3.

Table 3: K-Means and GMM Algorithm Validation Losses on 100D Dataset

K	**Validation Loss (K-Means)**	**Validation Loss (GMM)**
5	18443.85	289715.32
10	14239.45	80916.91
15	14222.35	77320.71
20	14114.49	76349.31
30	14115.80	76871.30

Using the data and patterns that occured in the 2D dataset, we can make an estimation on how many clusters exist within the 100D dataset. Through the pattern given in table 2, it shows that after it reaches the optimal cluster amount (in this case, k=3), the validation loss no longer drops significantly. Therefore, using this information, we can assume that the number of clusters that exists in this dataset is between 6 and 10 inclusive. The K-mean's loss values drop by 4000 when the number of clusters change from 5 to 10, and drop only by values less than 150 when increasing the number of clusters further. We obtain that the loss values have reached an optimal value as the loss does not drop significantly after this point. This is also shown through

the losses of GMM as the loss drops significantly from cluster numbers 5 to 10 showing that the optimal number of clusters is between 6 and 10 as the loss no longer drops significantly after this point. An optimal number of clusters have been reached.