# ECE421 Assignment 3 Report

## 1.1 Learning K-means

1. *Implementing distance_func() and running the K-means Algorithm for K=3 on data2D.npy. Some of the parameters used: iterations=200, use stddev = 0.5. K-means using the Adam Optimizer and Tensorflow 1 (assignments(X, mu) is a helper function)*

```
# 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, 0)
    mu = tf.expand_dims(mu, 1)
    pair_dist = tf.reduce_sum(tf.square(tf.subtract(X, mu)), 2)
    # returns KxN so transpose it because we need NxK
    return tf.transpose(pair_dist)
```

Figure 1: code for distance_func

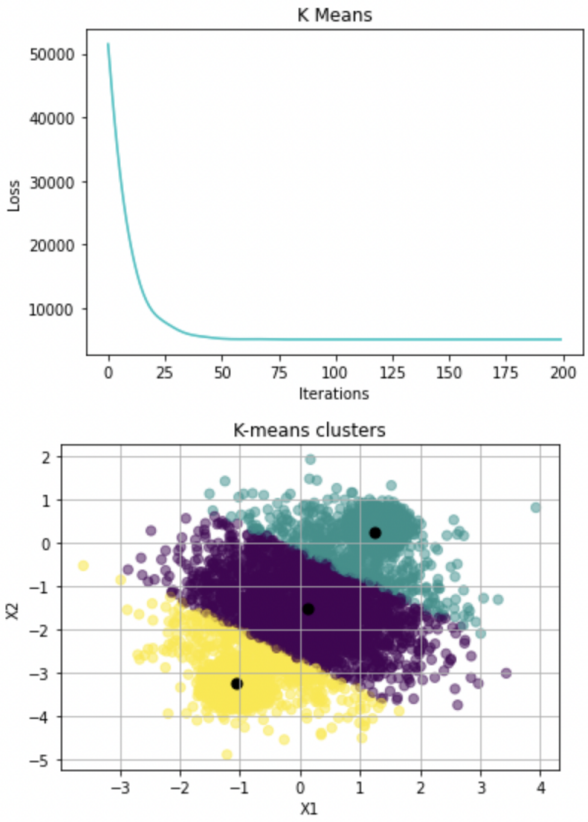



Figure 2: Loss and clusters for K=3

```
def k_means(K, is_valid):
  # Loading data
  data = np.load('data2D.npy')
  #data = np.load('data100D.npy')
  [num_pts, dim] = np.shape(data)

  if is_valid:
    valid_batch = int(num_pts / 3.0)
    np.random.seed(45689)
    rnd_idx = np.arange(num_pts)
    np.random.shuffle(rnd_idx)
    val_data = data[rnd_idx[:valid_batch]]
    data = data[rnd_idx[valid_batch:]]


  loss_array = []
  loss_array_val = []
  N = num_pts
  D = dim
  iterations = 200

  # create x as a placeholder (nonexD)
  X = tf.placeholder("float", shape=[None, D])
  #X = tf.constant(data, dtype=tf.float32)
  # need standard distributions values (KxD)
  mu_std = tf.truncated_normal([K, D], stddev=0.5)
  mu = tf.Variable(mu_std)

  # calculate distance between cluster centers and points
  # https://docs.w3cub.com/tensorflow~python/tf/reduce_sum
  # loss is minimizing the distance
  pair_dist = distance_func(X, mu)
  loss = tf.reduce_sum(tf.reduce_min(pair_dist, axis=1))

  # using adamOptimizer as stated in handout
  optimizer = tf.train.AdamOptimizer(learning_rate = 0.1, beta1 = 0.9, beta2 = 0.99, epsilon=1e-5).minimize(loss)

  # init globals and start the session
  globals_init = tf.global_variables_initializer()
  sess = tf.Session()
  sess.run(globals_init)

  # start training
  for i in range(iterations):
    # for validation data
    # set X=val_data and run session
    if is_valid:
      center_val, loss_curr_val, opt_val =  sess.run([mu, loss, optimizer], feed_dict={X:val_data})
      loss_array_val.append(loss_curr_val)
      # print info every 10 iterations
      if i%10 == 0:
        print("Iteration: ", i, " Loss: ", loss_curr_val)

    else:
      # for training data
      # set X=data and run session
      center, loss_curr, opt = sess.run([mu, loss, optimizer], feed_dict={X:data})
      loss_array.append(loss_curr)

      # print info every 10 iterations
      if i%10 == 0:
        print("Iteration: ", i, " Loss: ", loss_curr)

  assign_clusters = sess.run(assignments(X, mu), feed_dict={X:data, mu:center})
```

```
# this function finds the closest cluster center to a point and extracts it
def assignments(X, mu):
  dist = distance_func(X, mu)
  # extract the most small value from dist
  return tf.argmin(dist, 1)
```

Figure 3: Code for K-means for Training data and K=3

2. *K-means for Validation Data* for K={1, 2, 3, 4, 5}

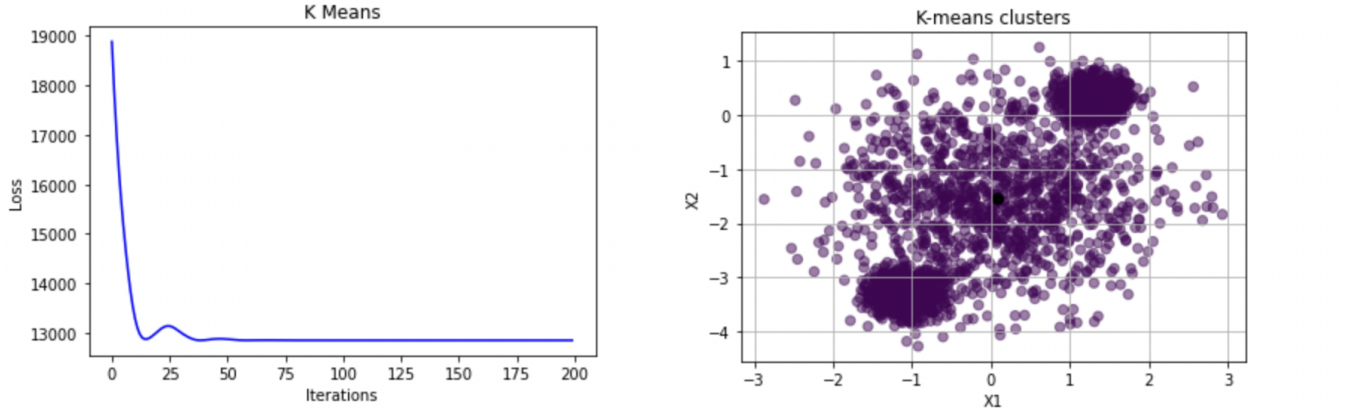



Figure 4: Clusters and Loss for K=1

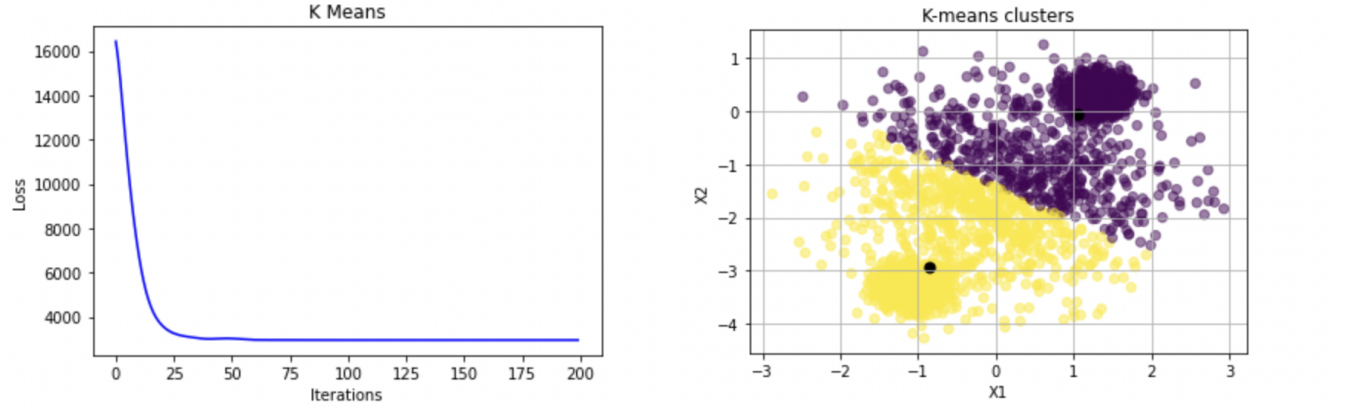



Figure 5: Clusters and Loss for K=2

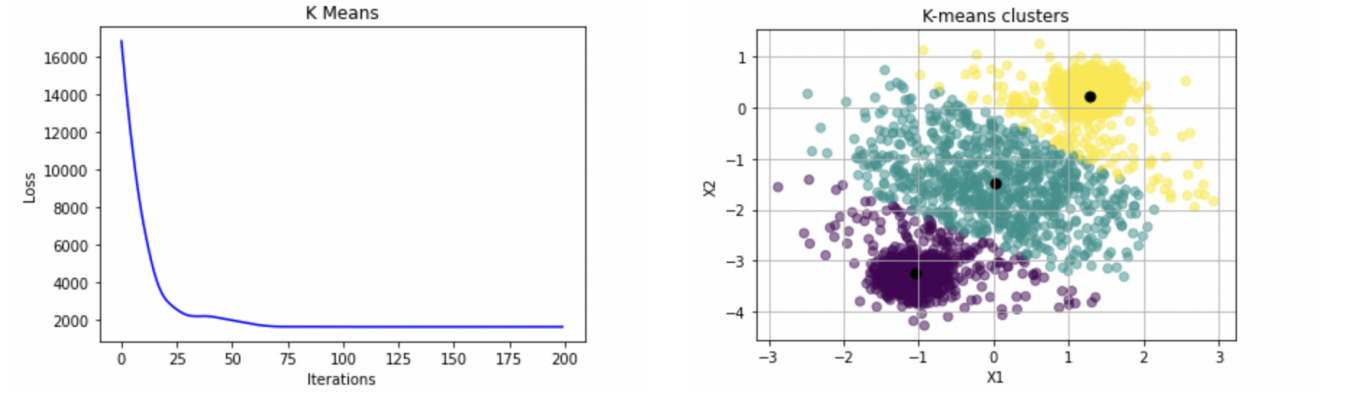



Figure 6: Clusters and Loss for K=3

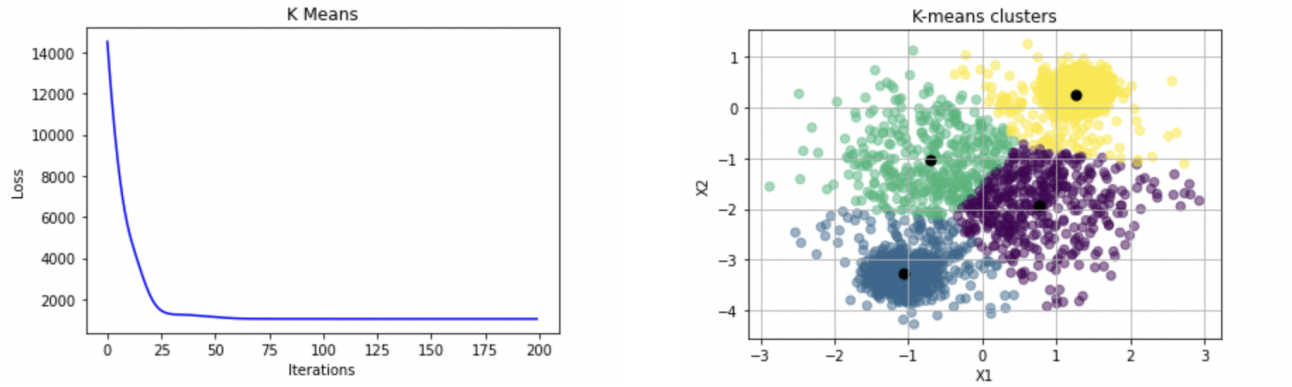



Figure 7: Clusters and Loss for K=4

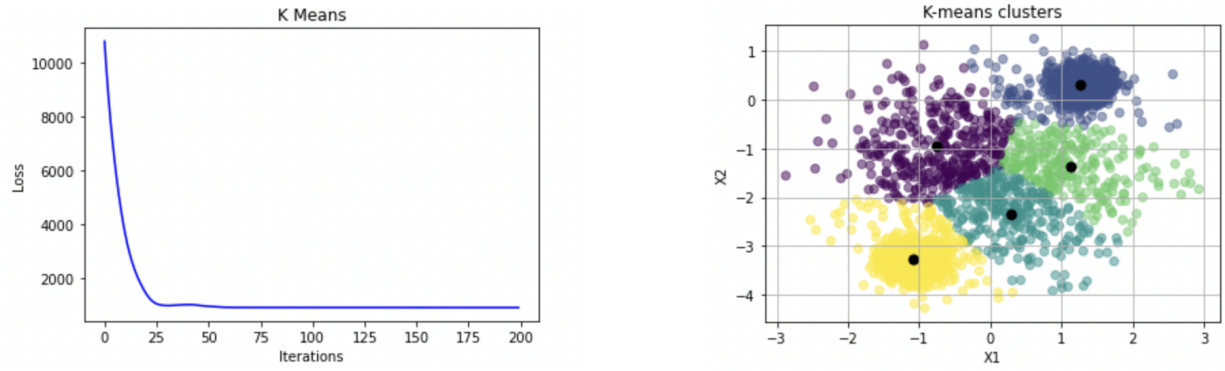



Figure 8: Clusters and Loss for K=5

Table 1: Shows the percentage of points belonging to a cluster with different K's

| K | % Class 1 | % Class 2 | % Class 3 | % Class 4 | % Class 5 |
|---|---|---|---|---|---|
| 1 | 100 | | | | |
| 2 | 51.82 | 48.18 | | | |
| 3 | 37.50 | 39.54 | 22.95 | | |
| 4 | 13.14 | 38.82 | 11.40 | 36.63 | |
| 5 | 10.62 | 7.71 | 38.34 | 6.96 | 36.36 |

Table 2: Shows the final loss values depending on number of clusters

| K (# clusters) | Validation Loss |
|---|---|
| 1 | 12853.26 |
| 2 | 2958.65 |
| 3 | 1608.84 |
| 4 | 1052.84 |
| 5 | 884.77 |

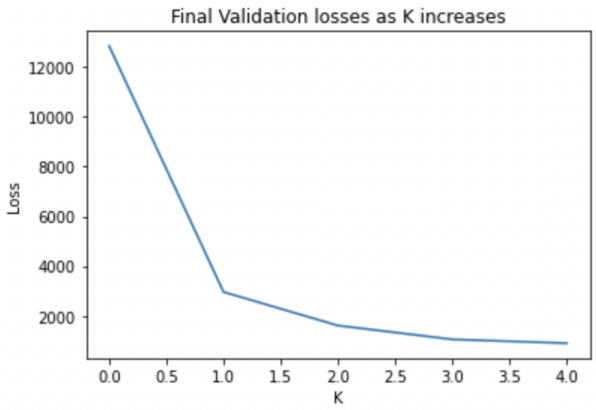



Figure 9: Validation losses as K changes

*By observing the percentages, we can see that for K=3, the distribution of points in the 3 clusters is more even compared to K=4, 5 where many data points are concentrated in one cluster. We can also see the loss starts to sort of level off at K=3 so the best choice is K=3.*

## 2.1 The Gaussian Cluster Mode

1. *Derived $\log P(x|\mu_k, \sigma_k^2)$ using the pdf equation for Gaussians. Derivation of all equations is attached in Appendix at the end of the report.*

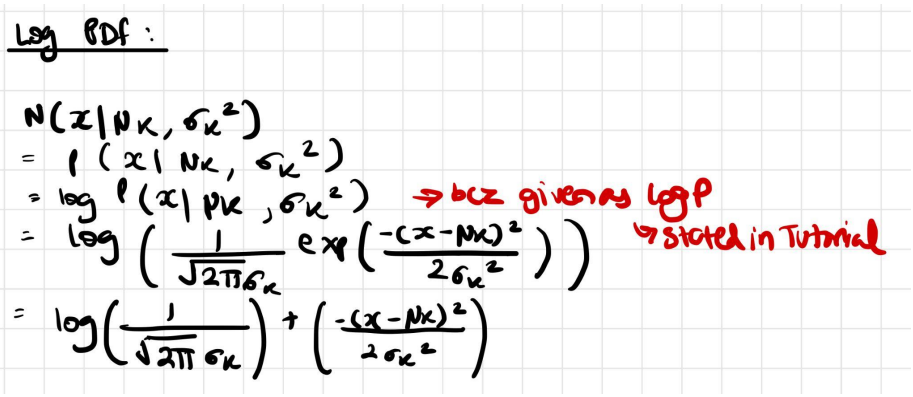



Figure 10: Derivation of $logP(x|\mu_k, \sigma_k^2)$

```
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)
    """
    x_mu = distance_func(X, mu)
    x_mu = tf.multiply(x_mu, x_mu)
    sigma = tf.squeeze(sigma)
    second_term = x_mu/(2*sigma)
    first_term = tf.log(2*np.pi*sigma) * tf.to_float(tf.rank(X))
    pdf = -0.5*first_term - second_term
    return pdf
```

Figure 11: Code for log_gauss_pdf()

2. *Derived logP(z|x) and wrote log_posterior(). It is important to use reduce_logsumexp() because it is more stable than using reduce_sum(). Taking exp of large values can cause overflows and exp of small numbers can cause underflows and this is handled well by the function.*

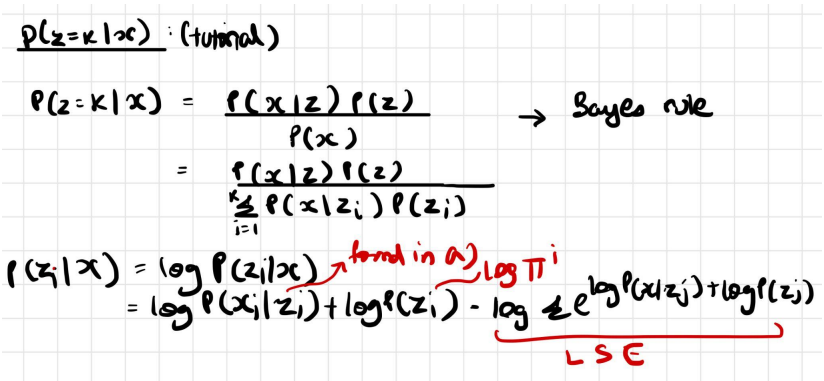



Figure 12: Derivation of $logP(z|x)$

```
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
    """
    logPI = tf.squeeze(log_pi)
    logP = tf.add(logPI, log_PDF)
    logSummation = hlp.reduce_logsumexp(log_PDF + logPI, keep_dims=True)
    final = logP - logSummation
    return final
```

Figure 13: Code for log_posterior()

## 2.2 Learning the MoG

1. *The model parameters that the algorithm has learnt are pi, mu, sigma which maximize $P(x_i)$ and minimize the log Loss.*

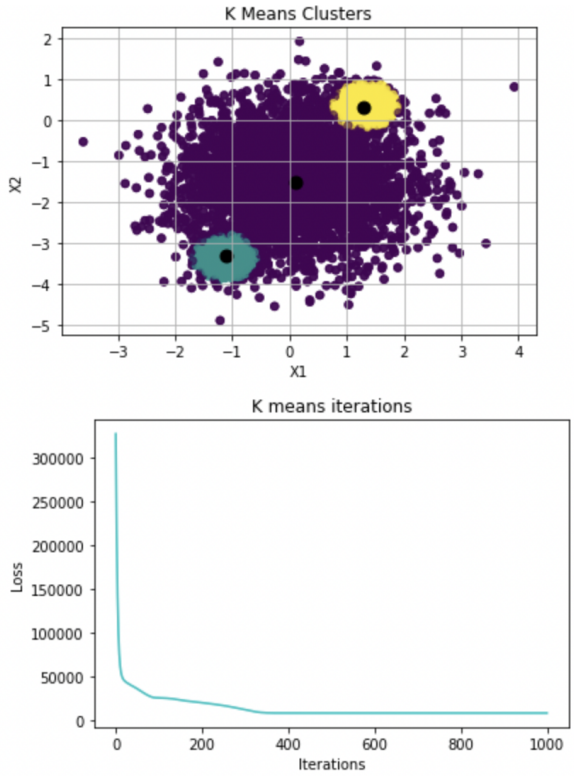



Figure 14: Clusters and Training loss for K=3

2. *Algorithm was run for K={1,2,3,4, 5}. The most reasonable choice for K is k=3. There was very little change in loss for K=4,5 and also for K=3 the data was more evenly distributed.*

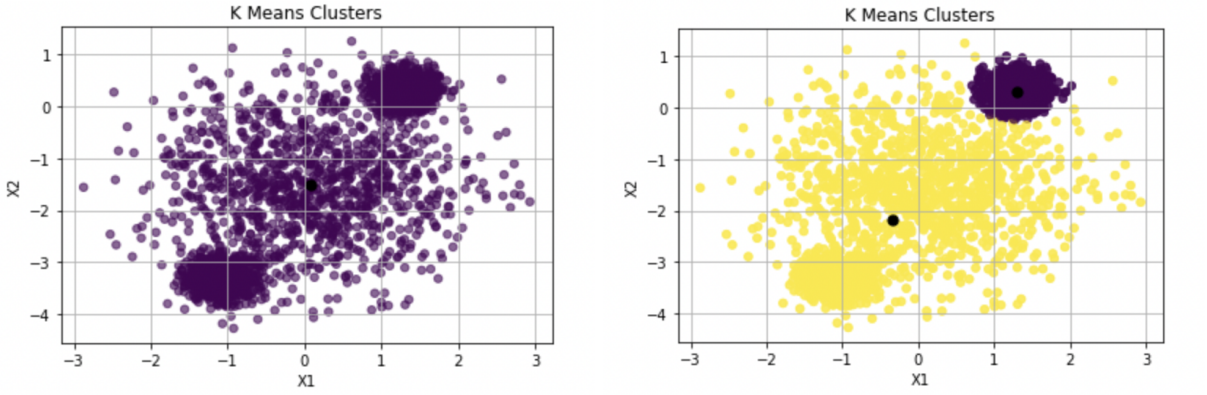



Figure 15: Clusters for K=1, 2

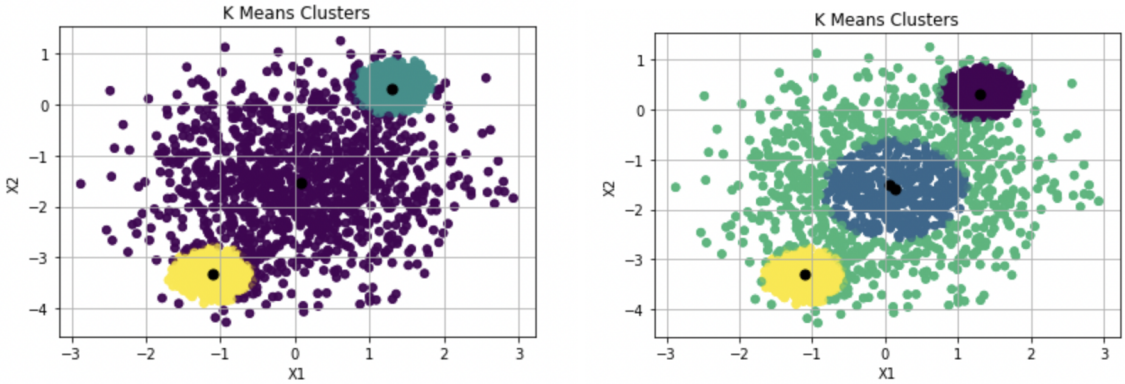



Figure 16: Clusters for K=3, 4

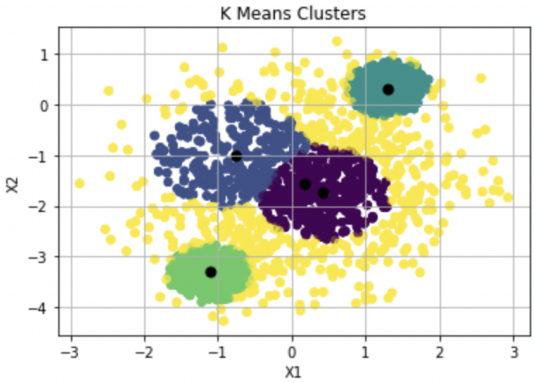



Figure 17: Clusters for K=5

Table 3: Shows the percentage of points belonging to a cluster with different K's

| K | % Class 1 | % Class 2 | % Class 3 | % Class 4 | % Class 5 |
|---|---|---|---|---|---|
| 1 | 100 | | | | |
| 2 | 33.60 | 66.40 | | | |
| 3 | 32.94 | 32.79 | 34.26 | | |
| 4 | 32.82 | 12.66 | 20.10 | 34.41 | |
| 5 | 9.84 | 8.07 | 32.97 | 34.47 | 14.64 |

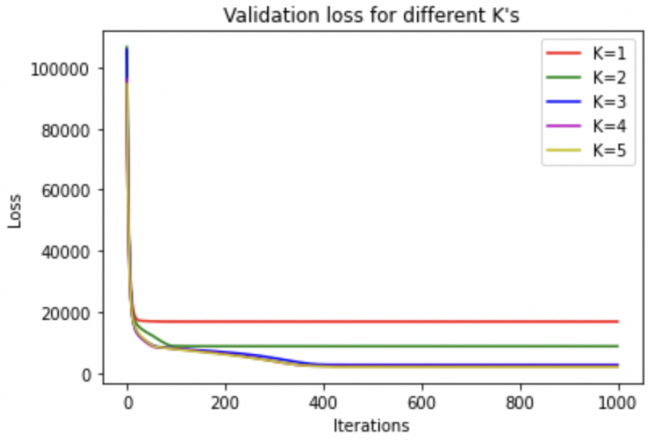



Figure 18: Validation Loss for different K values

Table 4: Shows the final loss values depending on number of clusters

| K (# clusters) | Validation Loss |
|---|---|
| 1 | 16815.32 |
| 2 | 8732.02 |
| 3 | 2678.17 |
| 4 | 2257.59 |
| 5 | 2009.33 |

3. *Many clusters past K=10, had 0% of points in some clusters for K=15,20,30. For the K-means, the good choice is around 10 clusters because there is very little change in loss for 15,20,30 and half the clusters are empty in those. However, we get lower overall losses using GMM in higher dimensions as we can see in Table 5. For GMM, around 15 clusters would be a good choice but the loss seemed to be fluctuating quite a bit, however it did have much better performance than K-means in higher dimensional data (data100D.py).*

Table 5: Shows the final loss values depending on number of clusters

| **K** | **5** | **10** | **15** | **20** | **30** |
|---|---|---|---|---|---|
| **K-means** | 71653.69 | 69918.68 | 68553.46 | 67972.11 | 67119.99 |
| **GMM** | 36884.13 | 37925.97 | 30505.32 | 34864.34 | 30573.30 |

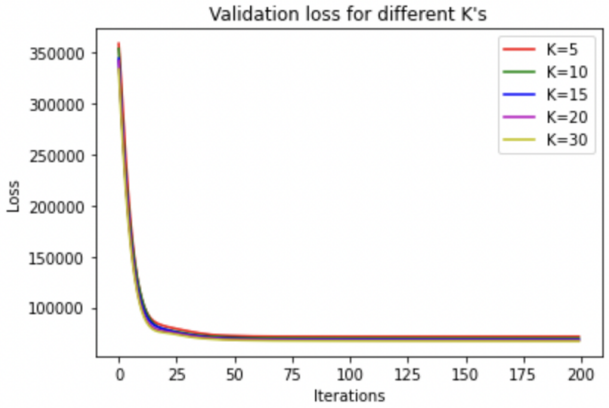



Figure 18: K-Means

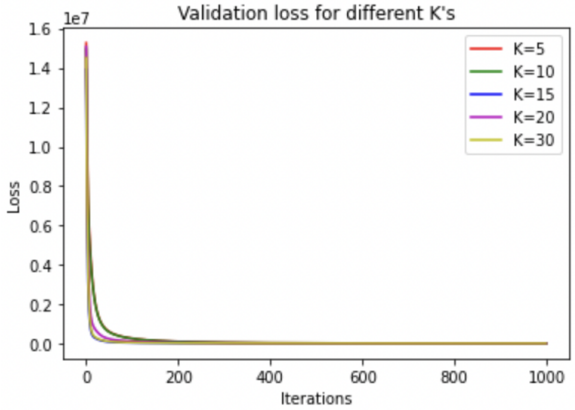
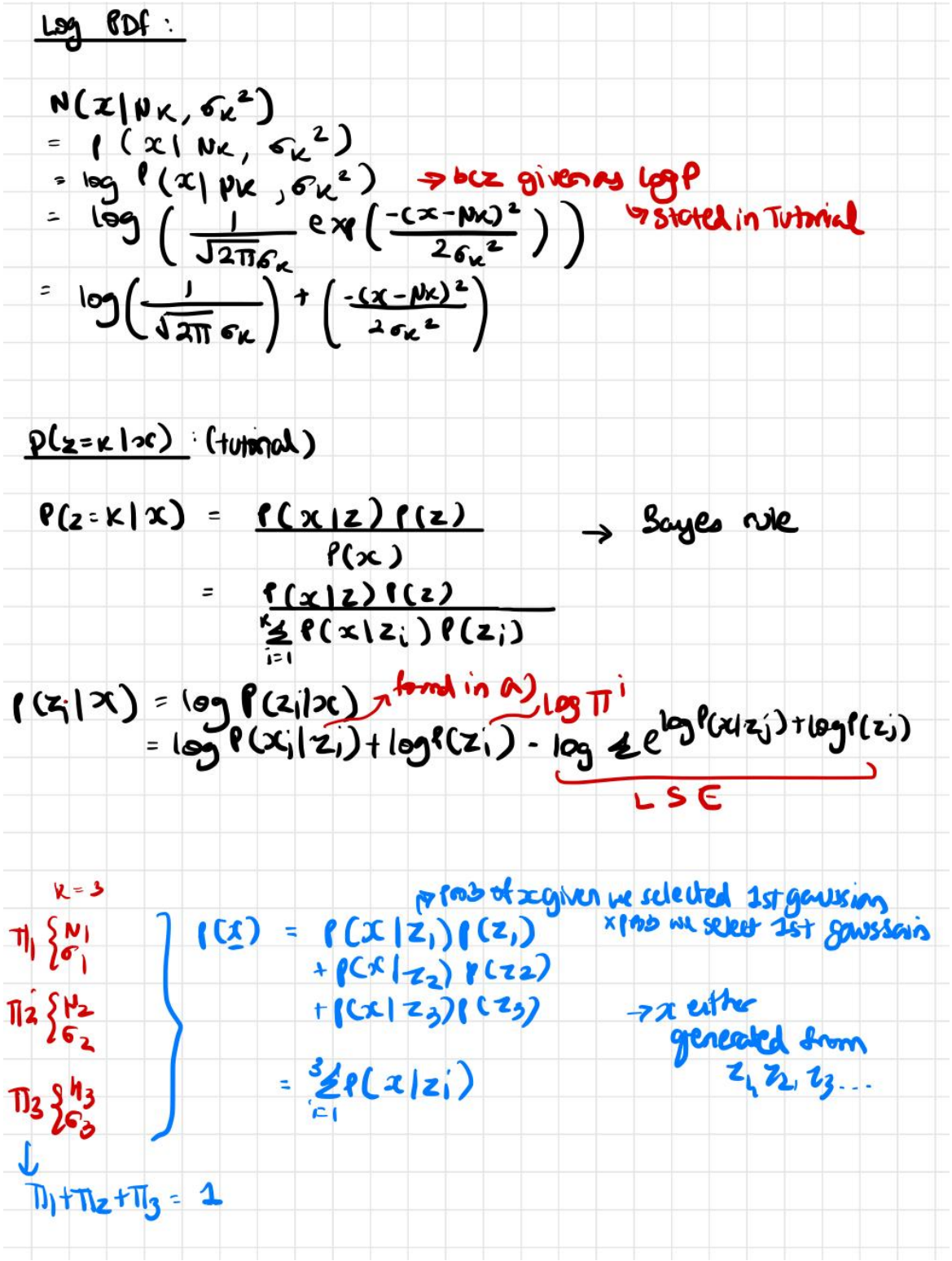



Figure 19: GMM

## 3.0 Appendix


Log PDF :
$N(x|\mu_k, \sigma_k^2)$
$= p(x|\mu_k, \sigma_k^2)$
$= \log p(x|\mu_k, \sigma_k^2)$ → bcz given as logP
$= \log\left(\frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)\right)$ ↳ stated in Tutorial
$= \log\left(\frac{1}{\sqrt{2\pi}\sigma_k}\right) + \left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)$
$p(z=k|x)$ : (tutorial)
$p(z=k|x) = \frac{p(x|z)p(z)}{p(x)}$ → Bayes rule
$= \frac{p(x|z)p(z)}{\sum_{i=1}^{K} p(x|z_i)p(z_i)}$
$p(z_i|x) = \log p(z_i|x)$ found in a)
$= \log p(x_i|z_i) + \log p(z_i) - \log \sum e^{\log p(x|z_j) + \log p(z_j)}$
LSE
K = 3
$\pi_1$ {$\mu_1$, $\sigma_1$}
$\pi_2$ {$\mu_2$, $\sigma_2$}
$\pi_3$ {$\mu_3$, $\sigma_3$}
$\pi_1 + \pi_2 + \pi_3 = 1$
$p(x) = p(x|z_1)p(z_1)$ ↳ prob of x given we selected 1st gaussian
$+ p(x|z_2)p(z_2)$ x prob we select 1st gaussian
$+ p(x|z_3)p(z_3)$
$= \sum_{i=1}^{3} p(x|z_i)$
→ x either generated from $z_1, z_2, z_3 \ldots$

want to maximize this $p(\underline{x})$

$\theta = \{\pi_k, \mu_k, \sigma_k^2\}_{k=1}^{K}$ (parameters)

↳ want setting that maximizes p.

↳ $\arg\max_{\theta} \prod_{i=1}^{N} p_\theta(\underline{x}_i)$ } training. ML

↳ $\underline{x}$ given an $\underline{x}$, want to know prob of it generated from 1, 2, 3

$$p(z|x) = \begin{bmatrix} p(z=1|x) \\ p(z=2|x) \\ p(z=3|x) \end{bmatrix}$$

$$p(z_i|x) = \frac{p(x|z_i)\,p(z_i)}{\sum_{j=1}^{K} p(x|z_j)\,p(z_j)}$$ Bayes

eg: $p(z=0|x) = \frac{p(x|z=0)\,p(z=0)}{p(x|z=0)\,p(z_0) + p(x|z=1)\,p(z_1)\ldots}$

↑ Inference

$$\arg\max_{\theta} \prod_{i=1}^{N} p_\theta(x_i)$$

$$\arg\max_{\theta} \sum \log p_\theta(x_i)$$

we have log of this not only p

$$= -\sum_{i=1}^{N} \log\left(\sum_{k=1}^{K} p(x_i|z=k)\,p(z=k)\right)$$

$p(x_i|z=k)$: pdf of gaussian; $p(z=k)$: $\pi_k$

$$= -\sum_{i=1}^{n} \log \sum_{k=1}^{K} e^{\log p(x_i \mid z=k) p(z=k)}$$

$$= -\sum_{i=1}^{n} \log \sum_{k=1}^{K} e^{\log p(x_i \mid z=k) + \log p(z=k)}$$

① $\quad \pi$

sum[LSE(①, π)]

U L U S U E

log_sum_exp()

↳ similar to softmax.

↳ in helper.py

$\log \sum e^0$

$$p(z_i \mid x) = \log p(z_i \mid x) \quad \text{(found in a)} \quad \log \pi^i$$

$$= \log p(x_i \mid z_i) + \log p(z_i) + \log \sum e^{\log p(x_i \mid z_j) + \log p(z_j)}$$

LSE

$\theta = \{\pi_k, \mu_k, \sigma_k^2\}_{k=1}^{K}$ (parameters)

$\psi_i \rightarrow$ use $\psi_i$ as parameter. $\rightarrow$ unconstraint $\pi$

$\underline{\pi} = \text{softmax}(\psi)$

use this in gradient descent.