1 K-means

1.1 Learning K-means

1.1.1 Distance Function

```
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)
    """
    broad_X = tf.expand_dims(X, axis=1)
    broad_mu = tf.expand_dims(mu, axis=0)

    return tf.reduce_sum(
        tf.square(tf.subtract(broad_X, broad_mu)),
        axis=-1
    )
```

The subtraction operation can be broadcasted by expanding different dimensions for X and μ. As provided by the hint in the assignment, both variables are expanded as such:

$$X' \in \mathcal{R}^{N \times 1 \times D}$$
$$\mu' \in \mathcal{R}^{1 \times K \times D}$$

The `tf.subtract()` function will then broadcast the subtraction operation across the first and second axes, producing the pairwise differences. These differences are squared and summed across the last axis D to produce a distance tensor $\delta \in \mathcal{R}^{N \times K}$.

1.1.2 Training without Validation

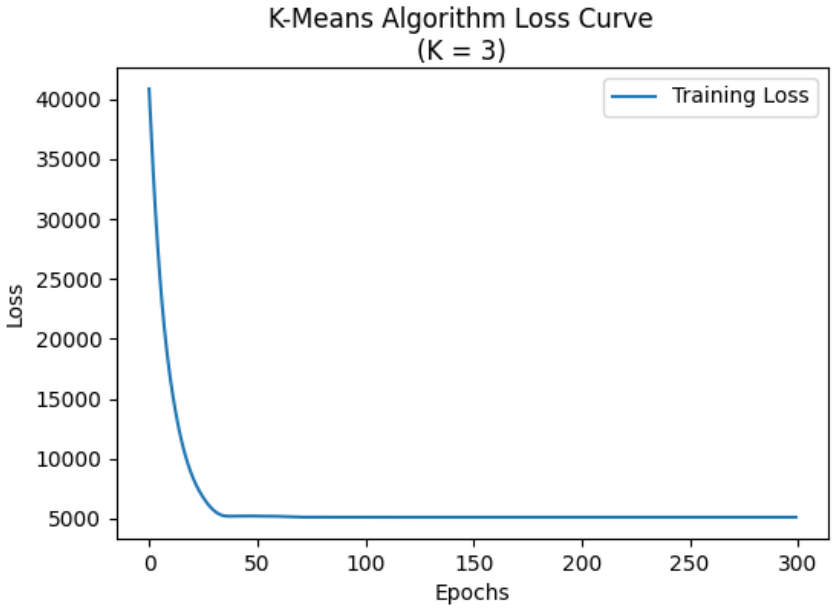

Figure 1: $K = 3$ with no validation dataset

1.1.3 Training with Validation

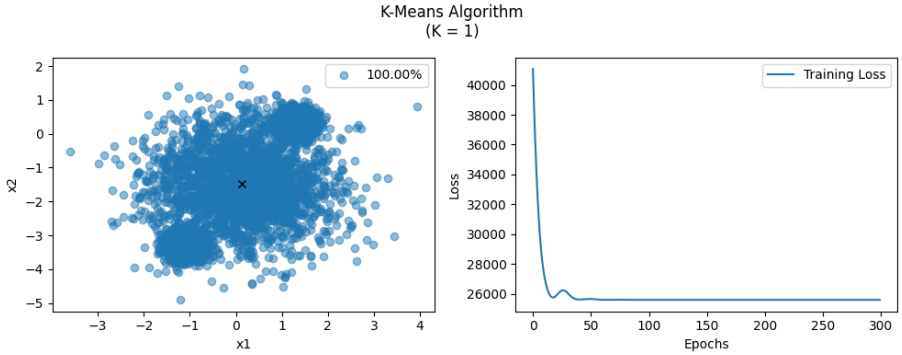

Figure 2: Validation loss = 12870.104

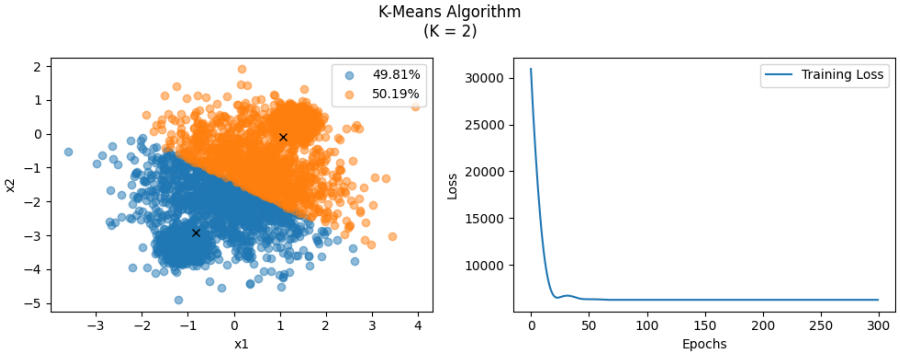

Figure 3: Validation loss = 2960.673

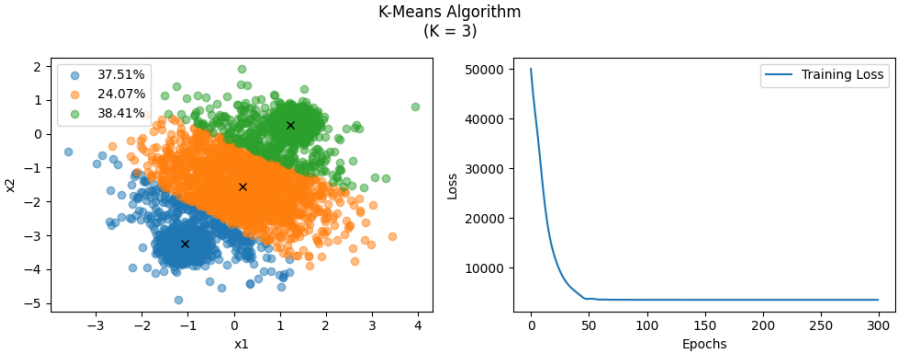

Figure 4: Validation loss = 1629.309

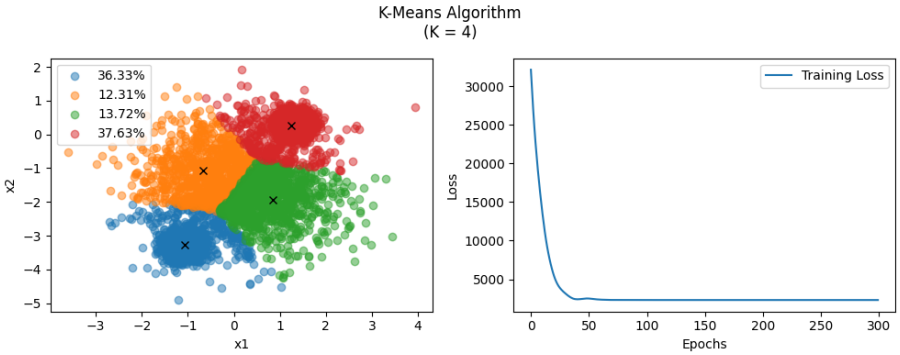

Figure 5: Validation loss = 1054.538

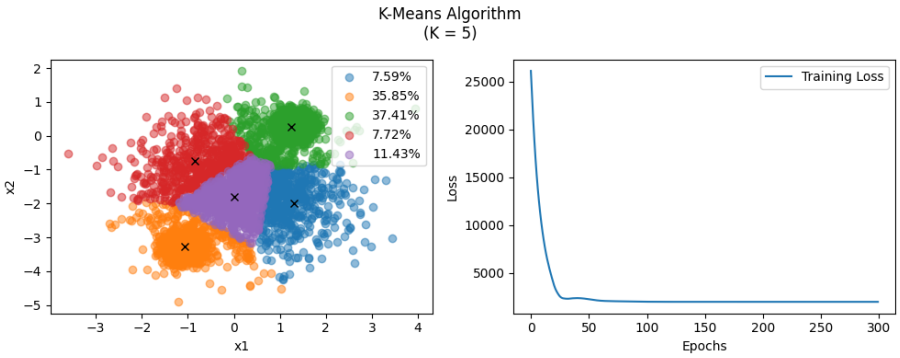

Figure 6: Validation loss = 900.777

For a seemingly uniformly-random dataset, as the number of clusters increase, there is a natural tendency for the clusters to maximize the distances between each other, in order to minimize the distances to neighbouring data points. It can be argued that as the number of clusters approach the number of data points, the validation loss will decrease, until it reaches a critical limit of $N = K$, at which point the validation loss will be 0 and the system is perfect. This behaviour is reflected in the gradually decreasing validation loss from figures 2 to 6. From the validation loss alone, the optimal value is $K = 5$, which could be "optimized" further by increasing K.

As a result, deciding on a value for K solely based on the minimum loss is misguided. By observing the general spread of data, there are two clear clusters around $(1.5, 0.5)$ and $(-1, -3.5)$, with one large cluster around $(0, -2)$. Without any additional information about the nature of data, the best estimation is $K = 3$.

2 Mixtures of Gaussians

2.1 The Gaussian Cluster Mode

2.1.1 Log Probability Density Function

$$
\begin{aligned}
\log P(\mathbf{x}|z=k) &= \log \mathcal{N}(\mathbf{x}; \mu_k, \sigma_k^2) \\
&= \log \left[\frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x_i-\mu_k)^T \Sigma^{-1}(x_i-\mu_k)} \right] \\
&= \log \left[\frac{1}{(2\pi)^{\frac{d}{2}} \sigma^d} e^{-\frac{1}{2}\frac{(x_i-\mu_k)^T(x_i-\mu_k)}{\sigma^2}} \right] \\
&= \log \left[\frac{1}{\left(\sqrt{2\pi}\sigma\right)^d} e^{-\frac{1}{2\sigma^2}(x_i-\mu_k)^2} \right] \\
&= -d \log \left(\sqrt{2\pi}\sigma\right) - \left(\frac{1}{2\sigma^2}(x_i-\mu_k)^2 \right) \qquad (1)
\end{aligned}
$$

In the final simplification shown in equation 1, the squared difference $(x_i - \mu_k)^2$ between the observation and the mean of the k^{th} cluster is equivalent to the Euclidean distance function for K-means, as shown in section 1.1.1.

```
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)
    """

    # sigma_T: 1 X K
    sigma_T = tf.transpose(sigma)

    distance = starter_kmeans.distance_func(X, mu)
    return tf.subtract(
        tf.multiply(-dim, tf.math.log(tf.sqrt(2 * np.pi) * sigma_T)),
        tf.divide(distance, (2 * tf.square(sigma_T)))
    )
```

It is important to note that since $\sigma \in \mathcal{R}^{K \times 1}$, it must be transposed to broadcast operations with ***`distance`*** *and produce an output shape of $N \times K$.*

2.1.2 Log Probability

$$
\begin{aligned}
\log P(z|\mathbf{x}) &= \log \left[\frac{P(\mathbf{x}, z=k)}{\sum_{j=1}^{K} P(\mathbf{x}, z=j)} \right] \\
&= \log \left[\frac{P(\mathbf{x}|z=k)P(z=k)}{\sum_{j=1}^{K} P(\mathbf{x}|z=j)P(z=j)} \right] \\
&= \log \left[\frac{P(\mathbf{x}|z=k)\pi_k}{\sum_{j=1}^{K} P(\mathbf{x}|z=j)\pi_j} \right] \\
&= \log P(\mathbf{x}|z=k) + \log \pi_k - \log \left[\sum_{j=1}^{K} P(\mathbf{x}|z=j)\pi_j \right] \qquad (2) \\
&= \log P(\mathbf{x}|z=k) + \log \pi_k - \log \left[\sum_{j=1}^{K} e^{\log P(\mathbf{x}|z=j)\pi_j} \right] \\
&= \log P(\mathbf{x}|z=k) + \log \pi_k - \log \left[\sum_{j=1}^{K} e^{\log P(\mathbf{x}|z=j) + \log \pi_j} \right] \qquad (3)
\end{aligned}
$$

Comparing equation 2 and 3, equation 2 may seem like the better choice as it is simpler. In terms of mathematical computation, however, equation 3 maximizes the amount of computational overlap for $\log P(\mathbf{x}|z=k)$ and $\log \pi_k$ by converting the probability within the summation into its logarithmic form. This allows reuse of logarithmic Gaussian probability density function in section 2.1.1 in conjunction with `reduce_sumlogexp()`, which applies a logarithm after the summation of exponents: $\log \sum e^x$. Calculating either equation with `tf.reduce_sum()` would require the calculation of $P(\mathbf{x}|z=j)$ or cost additional work.

```
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
    """

    # Avoid computational redundancy
    sum_log_PDF_pi = tf.add(log_PDF, tf.transpose(log_pi))
    return tf.subtract(
        sum_log_PDF_pi,
        hlp.reduce_logsumexp(sum_log_PDF_pi, keepdims=True)
    )
```

2.2 Learning the MoG

2.2.1 Negative Log Likelihood

$$
\begin{aligned}
\mathcal{L}(\mu, \sigma, \pi) &= -\log P(\mathbf{x}) \\
&= -\log \left[\sum_{j=1}^{K} P(\mathbf{x}|z=j) P(z=j) \right] \\
&= -\log \left[\sum_{j=1}^{K} P(\mathbf{x}|z=j) \pi_j \right] \\
&= -\log \left[\sum_{j=1}^{K} e^{\log P(\mathbf{x}|z=j) \pi_j} \right] \\
&= -\log \left[\sum_{j=1}^{K} e^{\log P(\mathbf{x}|z=j) + \log \pi_j} \right]
\end{aligned}
$$

2.2.2 Training without Validation

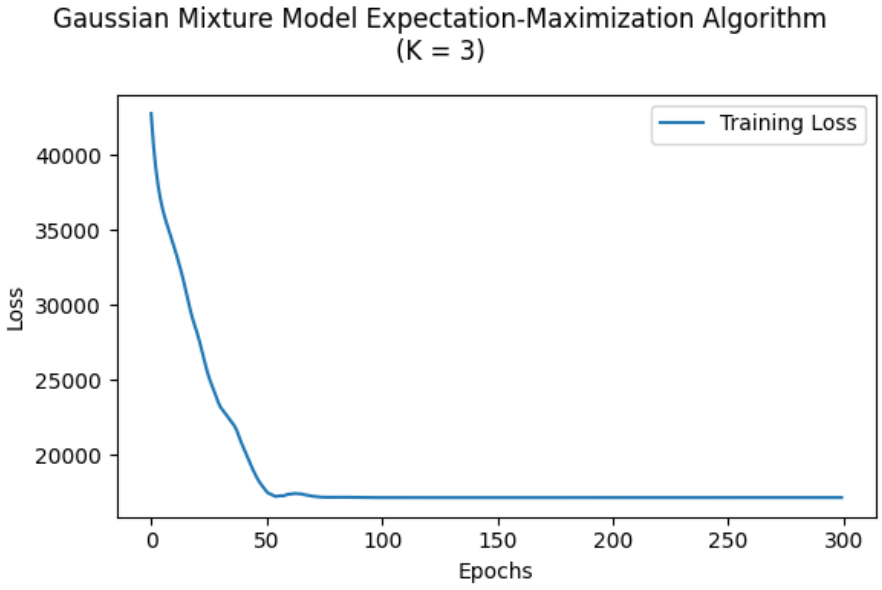

Figure 7: $K = 3$ with no validation dataset

2.2.3 Training with Validation

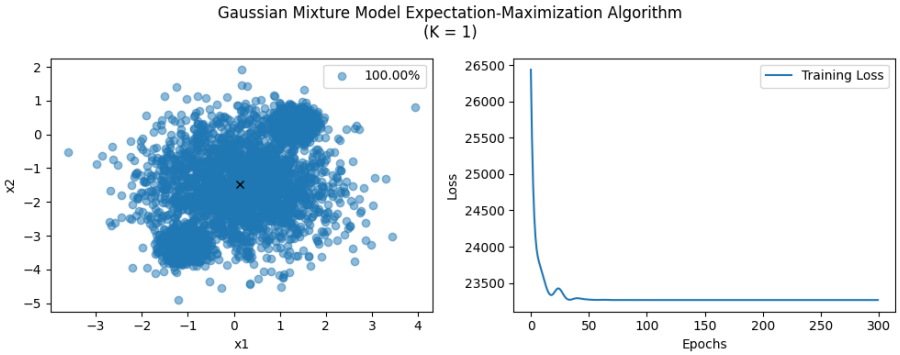

Figure 8: Validation loss = 11651.442

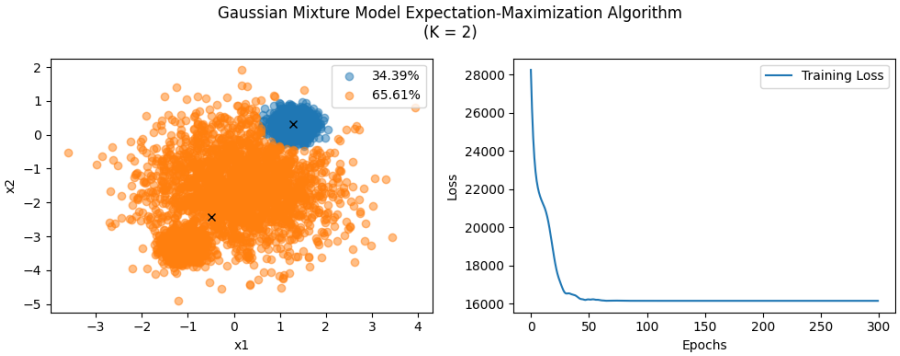

Figure 9: Validation loss = 7987.788

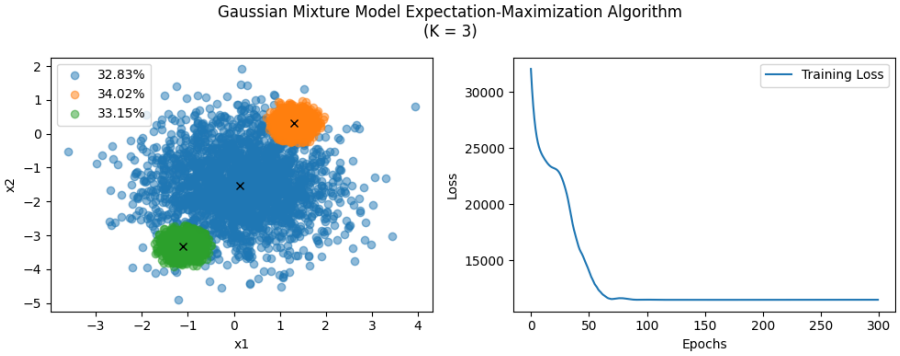

Figure 10: Validation loss = 5635.476

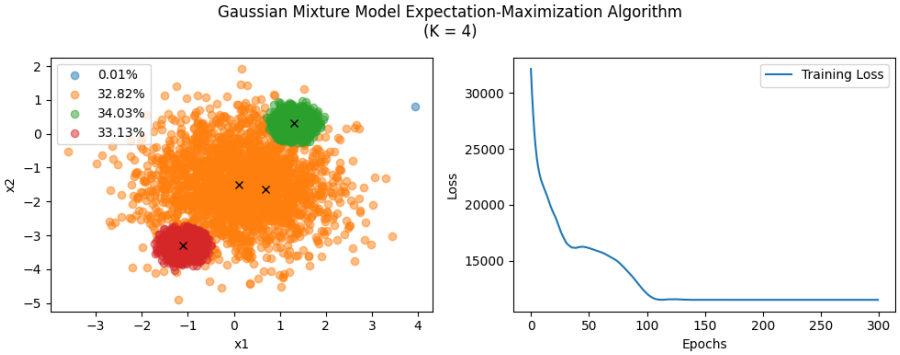

Figure 11: Validation loss = 5630.384

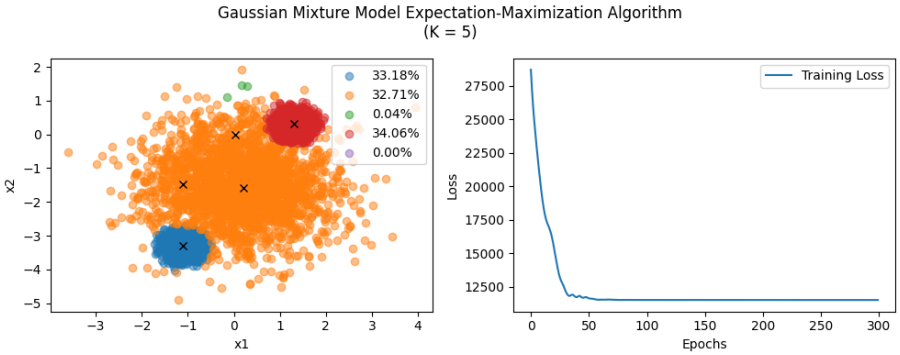

Figure 12: Validation loss = 5630.391

The cluster-loss issue with the K-means algorithm does not apply here. While K-means attempts to utilize every cluster to minimize the total loss, the expectation-maximization algorithm will "disregard" some clusters if they do not provide a large enough posterior probability of membership.

Noticeable from figure 10 onwards, as $K \geq 3$, the validation loss has a negligible increase as more clusters are added. The new cluster in figure 11 has a 0.01% data point membership, while the fifth cluster in figure 12 is completely unused. This indicates that it is most probable that there are a total of 3 Gaussian clusters in the model, and attempting to include more clusters than 3 will yield diminishing returns. Figure 10 will have the minimum theoretical validation loss, and thus the optimal number of clusters is $K = 3$.

2.3 `data100D.npy` Dataset (K-means vs. MoG Learning)

2.3.1 K-means

Cluster	Loss[1]	Maximum[2]	Median	Minimum
$K = 5$	121731.727	30.37%	20.04%	10.05%
$K = 10$	71230.969	29.29%	8.74%	0.00%
$K = 15$	69969.008	29.31%	3.00%	0.00%
$K = 20$	70461.008	20.35%	0.00%	0.00%
$K = 30$	68927.898	20.35%	1.31%	0.00%

2.3.2 MoG Learning

Cluster	Loss	Maximum	Median	Minimum
$K = 5$	272222.594	49.66%	20.04%	0.00%
$K = 10$	163250.906	29.31%	5.01%	0.00%
$K = 15$	163248.297	29.31%	0.00%	0.00%
$K = 20$	161845.266	20.35%	0.00%	0.00%
$K = 30$	162633.234	29.31%	0.00%	0.00%

2.3.3 Analysis

For both K-means and MoG learning, the validation loss plateaus and not all clusters are used from $K \geq 10$ onwards. This indicates that the optimal cluster structure can be reached with cluster counts of $K < 10$. Observing the zeroed median value for cluster $K = 15$ for MoG learning, an even tighter bound of $K \leq 15//2 = 8$ can be inferred. Since no more information can be extracted from the data collected, the estimation for the optimal number of clusters is $5 < K \leq 8$, or $K = 7$ to estimate further. Both MoG and K-means seem to reach similar cluster structures, both achieving a maximum membership percentage of $\approx 30\%$ in post-optimal conditions.

[1] Validation loss
[2] Values for the membership percentage of all clusters