Assignment 4

Machine Learning 1, SS24

Team Members		
Last name	First name	Matriculation Number
Nožić	Naida	12336462
Hinum-Wagner	Jakob	01430617

1 K-means. Expectation-Maximization Algorithm [25 points]

In this task, we will implement the K-means and Expectation-Maximization algorithms, and evaluate them using the *Mickey Mouse dataset*. Functions for loading the dataset, plotting the original data, and plotting the clusters are already provided.

K-means Algorithm

Suppose we are given a dataset $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}\}$ (with $\mathbf{x}^{(i)} \in \mathbb{R}^D$ for all $i \in \{1, \ldots, N\}$), we aim to partition the dataset into K clusters $\{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$. Each cluster $\mathcal{D}_k$ is represented by a *centroid vector* $\boldsymbol{\mu}^{(k)} \in \mathbb{R}^D$. Recall that the number of clusters K is a hyperparameter that we need to choose.

One approach to clustering is to solve the k-means problem, i.e., minimize the *within-cluster sum-of-squares* (WCSS):

$$\min_{\boldsymbol{\mu}^{(1)}, \ldots, \boldsymbol{\mu}^{(K)}, Z} \sum_{i=1}^{N} \sum_{k=1}^{K} z_{ik} \|\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(k)}\|_2^2 \tag{1}$$

with $z_{ik} \in \{0, 1\}$ and $\sum_k z_{ik} = 1$ for all $i \in \{1, \ldots, N\}$. Note that z_{ik} is a binary indicator variable that is 1 if datapoint $\mathbf{x}^{(i)}$ belongs to cluster k, and 0 else. More formally,

$$z_{ik} = \begin{cases} 1 & \text{if } k = \text{argmin}_j \, ||\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(j)}||_2^2 \\ 0 & \text{else} \end{cases} \tag{2}$$

This means, to each data point a one-hot vector of length K is assigned – there is exactly one 1 in the vector at the index of the assigned cluster, and the remaining entries of the vector are zeros.

The centroids (cluster means) are calculated as follows:

$$\boldsymbol{\mu}^{(k)} = \frac{1}{\sum_i z_{ik}} \sum_i z_{ik} \mathbf{x}^{(i)} \tag{3}$$

Tasks:

1.1 Tasks 1-6

The core steps of the K-means clustering algorithm is implemented through the first 6 tasks. The solution can be viewed in the provided code.

1.2 Task 7

The below images showcase the original, clustered dataset and the convergence behavior of the K-means algorithm. From simply observing the original data, we can notice three distinct clusters, with outliers in the outskirts. Therefore, as a starting point, we have set the K hyperparameter to 3 and max iterations to 100. With this being done, the plot now clearly organizes the data into three clusters, including the outliers whose cluster we beforehand could not easily identify. Additionally, the Figure 2 contains three centroids, which are centered in each cluster. In the Figure 3 we see that the algorithm quickly reduces the WCSS in the initial iterations. This indicates that the quality of the clustering has rapidly improved and that we have reached the plateau state at iteration 10. We found the local minimum of the objective function, since cluster assignments do not change much with further iterations.

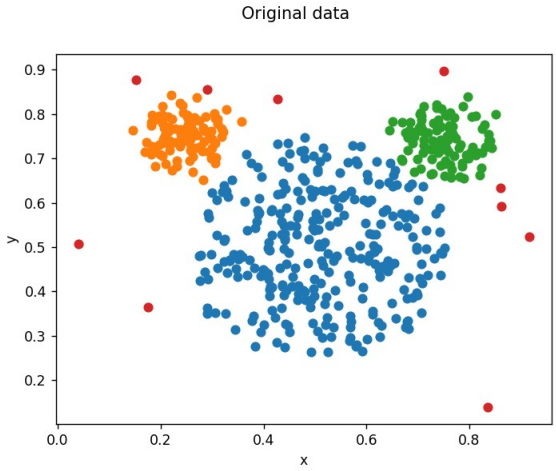

Figure 1: Plot of the original Mickey Mouse dataset

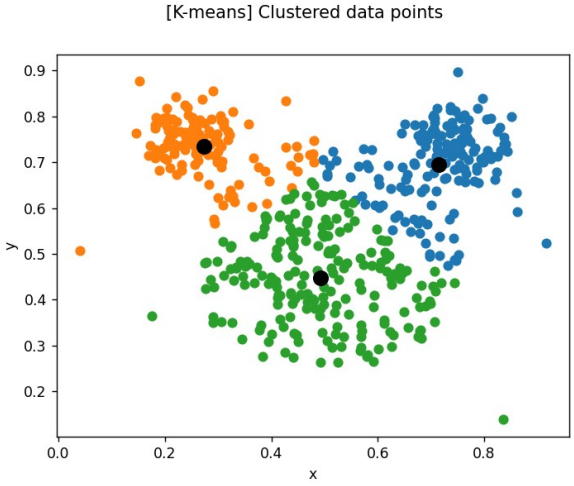

Figure 2: Mickey Mouse clustered data with k=3 and max_iter = 100

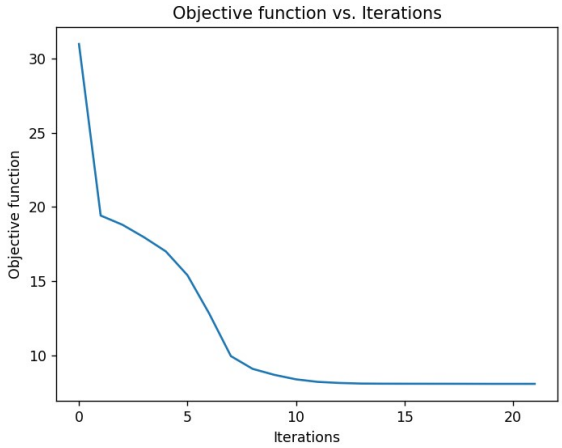

Figure 3: Mickey Mouse - Objective function vs Iterations (K=3)

1.3 Task 8

The new clustered dataset with K = 5 gives a more detailed segmentation compared to the previous plot with K = 3. The clusters are compact, spherical and approximately of equal sizes. The convergence is achieved also in the iteration 10. However, by looking at the plot Figure 5, we see a steep decline in the curve, but with less changes than in the K = 3, indicating a more stable and consistent improvement in clustering. Overall, choosing K = 5 as the final hyperparameter, seems to be a better choice due to the smooth, rapid convergence and comprehensive segmentation, which more accurately clusters the data points (especially the outliers).

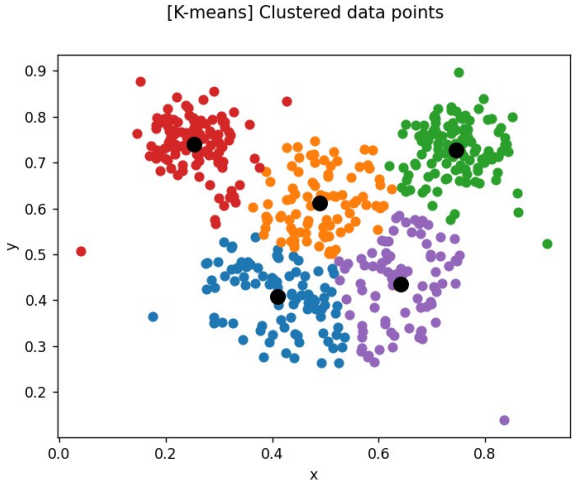

Figure 4: Mickey Mouse clustered data with k=5 and max_iter = 100

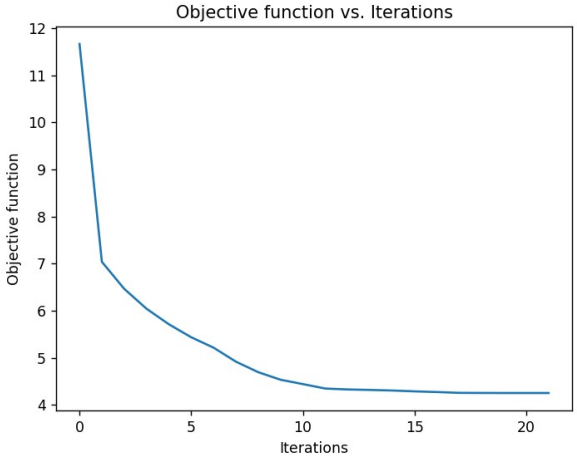

Figure 5: Mickey Mouse - Objective function vs Iterations (K=5)

1.4 Expectation-Maximization (EM) Algorithm [12 points]

A Gaussian Mixture Model (GMM) with K components is given as:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{k=1}^{K} w_k \, \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \Sigma_k) \tag{4}$$

with $\boldsymbol{\theta} = \{w_k, \boldsymbol{\mu}_k, \Sigma_k\}_{k=1}^{K}$ where w_k represent the component weights (with the constraints that $\sum_{k=1}^{K} w_k = 1$ and $0 \leq w_k$ for all $k \in \{1, \ldots, K\}$), $\boldsymbol{\mu}_k$ are the means, and Σ_k are the covariance matrices.

Given a dataset $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$, the steps of the EM algorithm are:

- Initialize the parameters $\boldsymbol{\theta} = \{w_k, \boldsymbol{\mu}_k, \Sigma_k\}_{k=1}^{K}$ using some initialization strategy.
- **Expectation step (E-step):** For each sample $\mathbf{x}^{(i)}$, calculate the cluster responsibilities (i.e., the probability that the sample was caused by the k-th component of the GMM):

$$\gamma_{ik} = \frac{w_k \mathcal{N}(\mathbf{x}^{(i)}; \boldsymbol{\mu}_k, \Sigma_k)}{\sum_{k'=1}^{K} w_{k'} \mathcal{N}(\mathbf{x}^{(i)}; \boldsymbol{\mu}_{k'}, \Sigma_{k'})} \tag{5}$$

  We can think of this as a soft version of Equation 2.

- **Maximization step (M-step):** Given the soft cluster assignments γ_{ik}, we can update $\boldsymbol{\theta}$:

$$\boldsymbol{\mu}_k \leftarrow \frac{1}{\sum_{i=1}^{N} \gamma_{ik}} \sum_{i=1}^{N} \gamma_{ik} \mathbf{x}^{(i)} \tag{6}$$

$$\Sigma_k \leftarrow \frac{1}{\sum_{i=1}^{N} \gamma_{ik}} \sum_{i=1}^{N} \gamma_{ik} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_k)(\mathbf{x}^{(i)} - \boldsymbol{\mu}_k)^T \tag{7}$$

$$w_k \leftarrow \frac{1}{N} \sum_{i=1}^{N} \gamma_{ik} \tag{8}$$

- Evaluate the log-likelihood function using the new parameters $\boldsymbol{\theta}$:

$$\log\left(p_{\boldsymbol{\theta}}(\mathcal{D})\right) = \sum_{i=1}^{N} \log\left(\sum_{k=1}^{K} w_k \, \mathcal{N}(\mathbf{x}^{(i)}; \boldsymbol{\mu}_k, \Sigma_k)\right) \tag{9}$$

  and check if the log-likelihood is the same as in the previous step (up to a small constant $\varepsilon > 0$). If this is the case, the algorithm has converged. Otherwise, repeat the EM steps until convergence.

1.5 Task 1

In the code (file `em.py`), in the function **`calculate_responsibilities()`** implement the E-Step of EM algorithm, namely, Equation 5 that calculates responsibilities γ_{ik} for each sample $\mathbf{x}^{(i)} \in \mathcal{D}$ and each cluster $k \in \{1, \ldots, K\}$. **Solution** In this task, the responsibilities for the E-step of the EM algorithm are calculated. The process begins by determining the number of data points N and the number of components K from the shapes of the input data matrix X and the means matrix `means`. A zero matrix for responsibilities is then initialized to store the values of γ_{ik}.

Next, the numerator of γ_{ik} for each data point and component is calculated. This is achieved by looping over each component k and computing the weighted probability density for each data point under the k-th Gaussian component using the `multivariate_normal.pdf` function from the `scipy.stats` library. The results are stored in the responsibilities matrix.

Following the calculation of the numerators, the denominator for each data point is computed. This is done by summing the values of the numerators across all components for each data point, resulting in a column vector of sums.

To obtain the final responsibilities, the numerators are normalized by dividing each element in the responsibilities matrix by the corresponding sum of the denominators. This ensures that the responsibilities for each data point sum to one.

Finally, the normalized responsibilities matrix is returned, completing the calculation for the E-step of the EM algorithm.

1.6 Task 2

In the code (file `em.py`), in the function **update_parameters()** implement the M-Step of EM algorithm, that is, Equation 6, Equation 7, and Equation 8.

Solution In this task, the parameters for the M-step of the EM algorithm are updated. The process begins by determining the number of data points N and the number of components K from the shapes of the input data matrix X and the means matrix `means`. Zero matrices for the updated means `means_new` and covariances `sigmas_new` are then initialized, as well as a zero vector for the updated weights `weights_new`.

Next, the new means $\boldsymbol{\mu}_k$ are calculated for each component k. This is achieved by summing the product of the responsibilities γ_{ik} and the data points $\mathbf{x}^{(i)}$, and then normalizing by the sum of the responsibilities for each component. This ensures that the new means are weighted by the responsibilities.

Following the calculation of the new means, the new covariances Σ_k are computed. For each component k, the difference between each data point $\mathbf{x}^{(i)}$ and the new mean $\boldsymbol{\mu}_k$ is calculated. The outer product of these differences, weighted by the responsibilities, is summed and then normalized by the sum of the responsibilities for each component. This results in the new covariance matrices, reflecting the spread of the data points around the new means.

To obtain the new weights w_k, the sum of the responsibilities for each component is computed and then normalized by the total number of data points N. This ensures that the weights represent the proportion of data points assigned to each component.

Finally, the updated means, covariances, and weights are returned, completing the M-step of the EM algorithm.

1.7 Task 3

In the code (file `em.py`), in the function **em()** implement the steps of EM algorithms that are missing. A strategy for initializing $\boldsymbol{\theta}$ is already provided.

Solution

In this task, the EM algorithm for Gaussian Mixture Models is finalized. The process begins by determining the number of data points N and the number of dimensions D from the input data matrix X. The number of components K is specified as a parameter.

Initialization: The parameters $\boldsymbol{\theta} = \{w_k, \boldsymbol{\mu}_k, \Sigma_k\}_{k=1}^{K}$ are initialized. Random initial means are generated, covariance matrices are initialized to be isotropic with a specified initial variance, and component weights are initialized equally.

Iterative Steps: A loop runs for a maximum number of iterations or until convergence. Within each iteration, the following steps are performed:

1. **Expectation Step (E-step):** Responsibilities γ_{ik} are calculated for each sample $\mathbf{x}^{(i)}$ using the `calculate_responsibilities` function. This function computes the probability that each data point belongs to each component of the GMM.
2. **Maximization Step (M-step):** The parameters $\boldsymbol{\theta}$ are updated using the `update_parameters` function. This involves recalculating the means, covariances, and weights based on the responsibilities computed in the E-step.
3. **Log-Likelihood Evaluation:** The log-likelihood of the data under the current model is evaluated

to check for convergence. The log-likelihood is computed as:

$$\log\left(p_{\boldsymbol{\theta}}(\mathcal{D})\right) = \sum_{i=1}^{N} \log\left(\sum_{k=1}^{K} w_k \, \mathcal{N}(\mathbf{x}^{(i)}; \boldsymbol{\mu}_k, \Sigma_k)\right)$$

4. **Convergence Check:** The algorithm checks if the change in log-likelihood is less than a small threshold ε. If this condition is met, the algorithm has converged, and the loop breaks.

Final Output: After convergence or reaching the maximum number of iterations, the function returns the final means, the soft assignments of data points to clusters, and the log-likelihood values over the iterations.

1.8 Task 4

In **task_em()** (file `main.py`), set K to the value that you used in K-means (in order to compare it with the results from the previous task), and choose an appropriate value for `max_iter`. The function **plot_objective_function()** will plot the log-likelihood over the iterations. Include the plot in the report.

Solution

We have tested out the same parameter values as in the K means algorithm, for better comparison. More specifically, we used K = 3 with max iterations set to 100. The corresponding graphs are provided below.

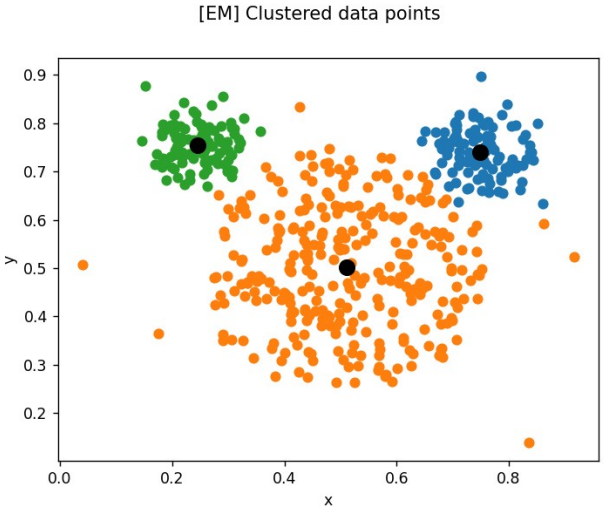

Figure 6: Mickey Mouse - clustered data with k = 3 and `max_iter` = 100

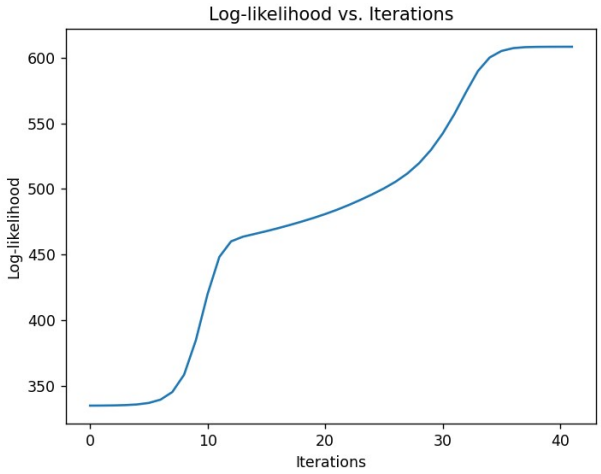

Figure 7: Mickey Mouse - Log-likelihood vs. Iterations with k = 3 and `max_iter` = 100

1.9 Task 5

There is already a function call to plot the Mickey Mouse after clustering. Include the plot in the report. Compare it with the plot of the original data (in one or two sentences). Include in the report the initial values of $(w_1, \ldots, w_K)$ and the final values (after the algorithm converged).

Solution
Below we have plots with the clustering algorithm applied with K = 5 and max iterations set to 100. Compared to the original data, the clustering algorithm was able to detect 5 clusters. It accurately handled the overlapping clusters, even better than K-means (the ears and head of the Mickey Mouse). Additionally, the three clusters in the head of the Mickey Mouse show more spread and overlap, reflecting the probabilistic nature of the algorithm which allows for softer cluster boundaries. In the K-means results, the clusters were more compact and of the same size approximately.

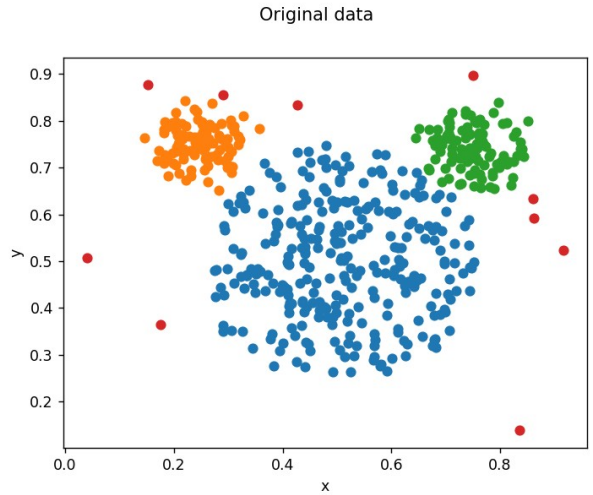

Figure 8: Mickey Mouse - original data

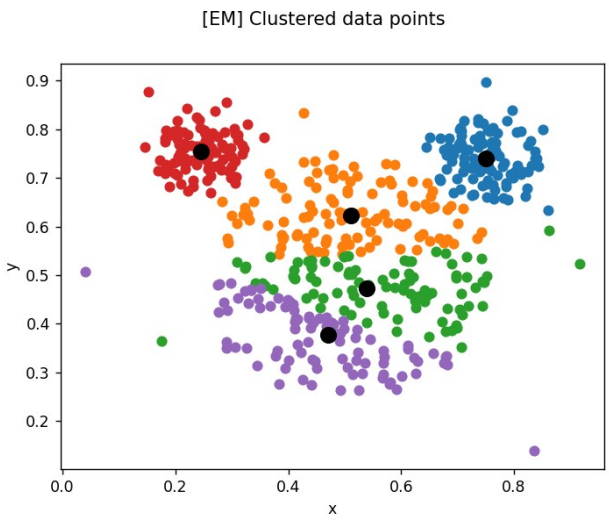

Figure 9: Mickey Mouse - clustered data with k = 5 and `max_iter` = 100

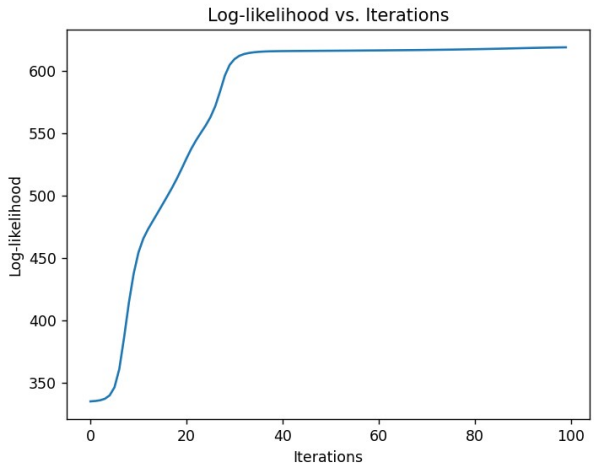

Figure 10: Mickey Mouse - Log-likelihood vs. Iterations with k = 5 and `max_iter` = 100

The initial weight values are:

`Weights: [0.2, 0.2, 0.2, 0.2, 0.2]`

After running the algorithm the weights are:

`Weights: [0.20017096, 0.21999307, 0.22399877, 0.19791548, 0.15792172]`

The initial means and covariances are:

```
Means:
[[0.75608784, 0.14273979],
 [0.0995017 , 0.96662246],
 [0.39148851, 0.90951968],
 [0.01294245, 0.03674949],
 [0.2955789 , 0.90567076]]

Covariances:
```

```
[[[1.5, 0.0 ],
  [0.0 , 1.5]],
 [[1.5, 0.0 ],
  [0.0 , 1.5]],
 [[1.5, 0.0 ],
  [0.0 , 1.5]],
 [[1.5, 0.0 ],
  [0.0 , 1.5]],
 [[1.5, 0.0 ],
  [0.0 , 1.5]]]
```

After running the algorithms, the new values are:

```
Means:
[[0.74976042, 0.74084045],
 [0.51032372, 0.6218799 ],
 [0.53903898, 0.47270102],
 [0.24521268, 0.75404209],
 [0.47028884, 0.37664755]]

Covariances:
[[[ 2.37885656e-03, -1.99220021e-04],
  [-1.99220021e-04,  2.37056622e-03]],
 [[ 1.49379956e-02, -8.22981928e-04],
  [-8.22981928e-04,  5.55461307e-03]],
 [[ 1.92910946e-02,  6.02007765e-04],
  [ 6.02007765e-04,  5.95618101e-03]],
 [[ 1.90534668e-03, -3.45336158e-05],
  [-3.45336158e-05,  1.68989963e-03]],
 [[ 1.54174125e-02, -5.10533367e-03],
  [-5.10533367e-03,  5.77180726e-03]]]
```

1.10 Summary and comparison of two algorithms [5 points]

Tasks:

1. Which algorithm works better for the Mouse data set? Why? Explain by comparing the plots of the original data, after K-means clustering, and after EM clustering using the GMM.

   Answer
   K means is a simple algorithm that assigns each data point to the nearest cluster. It assumes clusters are spherical and of equal sizes, which in this case does not accurately capture the true data structure. It also assumes the clusters are well balanced, which means that the k-means algorithm could fail if we have data clusters with far fewer observations (smaller density).

   If we observe the data with K = 3, we can notice that the GMM (EM) provides more accurate results. The reason why k - means is not the optimal choice is because of the fact that we have three unbalanced blobs of different sizes. The GMM is more flexible in this case, since it is able to capture different shapes and not only convex ones (e.g. a circle, a sphere), which is present in k-means.

   Furthermore, the probabilistic approach is overall better when we have overlapping clusters. In the specific dataset, the ears of Mickey Mouse (two smaller clusters) overlap slightly with the head cluster. K means would have issues here due to the fact that it assigns each data point to exactly one cluster. Having this said, the EM clustering using the GMM is the better algorithm for the Mickey Mouse dataset. It captures the mean location, standard deviation of the data and therefore is able to acquire the spatial extent of each cluster. K-means only knows the central location of clusters—nothing more.

2. Are there noisy data points (outliers) in the original data? Is K-means robust to noise? Is EM robust to noise?
Answer
There are outliers in the dataset, which is an additional reason as to why K-means is not the optimal algorithm choice for the Mickey Mouse dataset. K-means assigns data points to the nearest cluster centroid and aims to minimize the sum of squared distances. Having outliers can negatively effect the accuracy of the detected clusters, since they can pull the centroid away from the dense regions. On the other hand, GMM (EM) is not sensitive to outliers and noise because it assigns probabilities to data points belonging to each cluster instead of hard assigning them to a single cluster.

3. What is **the main difference** between the K-means and EM algorithm? Briefly discuss the connection between z_{ik} (K-means) and γ_{ik} (in EM).
Answer

The main difference between the two algorithms is in the way they assign data points to clusters. More specifically, K-means is characterized by using hard assignments (z_{ik}), while GMM (EM) uses soft assignments (γ_{ik}). The K-means algorithm iterates between two main steps. It first randomly initializes cluster centers, after which, for the given centers, computes cluster assignments for all data points. The main downside of this clustering is in the fact that each data point can only belong to one cluster defined by z_{ik} (1 for belonging to a cluster and 0 for not belonging).

$$z_{ik} = \begin{cases} 1 & \text{if } k = \text{argmin}_j \, ||\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(j)}||_2^2 \\ 0 & \text{else} \end{cases} \tag{10}$$

The centroids can be updated with the formula:

$$\boldsymbol{\mu}^{(k)} = \frac{1}{\sum_i z_{ik}} \sum_i z_{ik} \mathbf{x}^{(i)} \tag{11}$$

Regarding the GMM(EM), we use soft assignments, which means that we calculate probabilities of each data point belonging to each cluster. This is represented with the γ_{ik} value, calculated with:

$$\gamma_{ik} = \frac{w_k \mathcal{N}(\mathbf{x}^{(i)}; \boldsymbol{\mu}_k, \Sigma_k)}{\sum_{k'=1}^{K} w_{k'} \mathcal{N}(\mathbf{x}^{(i)}; \boldsymbol{\mu}_{k'}, \Sigma_{k'})} \tag{12}$$

We can then use the responsibilities to update the parameters (means, covariances, and priors) in the maximization Step (M-step). The main advantage of using soft assignments is in the fact that it makes the GMM(EM) more flexible in handling different cluster shapes, capturing complex patterns in the data, makes it more robust against outliers since they are less likely to significantly affect the parameters of the Gaussian distributions in GMM.

4. In the EM algorithm, the covariance matrices are updated in each iteration. However, the K-means algorithm does not have these parameters. What shape of clusters does K-means tend to produce? Hence, what are the assumed (implicit) covariance matrices for clusters in K-means?
Answer
K-means algorithm tends to produce clusters spherical in shape. This is due to the fact it uses Eucledian distance for assigning data points to the nearest centroid. Because of this, K-means works under the assumption that the variance of all clusters is the same in all directions. The density of data points, the size of the clusters is approximately the same and the shape is spherical.
Having this said, if we are working with a spherical shape, then the covariance matrix becomes a diagonal matrix with equal values along the diagonal. Elements in the diagonal represent the variances in each dimension and with them being equal we are capturing the property of K-means, which is the fact that variance of all clusters is the same in all directions.

5. In Task 1.2, we have learned parameters of a Gaussian Mixture Model, which we used to generate soft cluster assignments. Learning parameters of a probability distribution over our training data is called

density estimation and the resulting model can do more than just soft clustering our data points: For example, it is common to use such distributions to draw **samples** from this density (this is called **generative modeling**). Thus, assume we wish to sample a new data point $\mathbf{x}$ from our trained GMM $p_{\boldsymbol{\theta}}$. Briefly explain how we can interpret the component weights $(w_1, \ldots, w_K)$ as prior probabilities over a categorical latent variable and how this interpretation can be used to sample from $p_{\boldsymbol{\theta}}$. Write down the definition of $p_{\boldsymbol{\theta}}$ using this interpretation and state all steps needed to draw a sample from $p_{\boldsymbol{\theta}}$.

Answer

We can interpret the mixture weights $(w_1, \ldots, w_K)$ as probabilities of a categorical latent random dicrete variable Z that assumes values in some set $\{1, \ldots, K\}$, where latent means that it is not available in our training dataset. In other words, the weight (w_k) represents the probability of selecting a Gaussian component k in the mixture model. This can be mathematically represented as:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{k=1}^{K} w_k \, \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \Sigma_k) = \sum_{k=1}^{K} p(z = k) p(\mathbf{x}; \boldsymbol{\mu}_k, \Sigma_k) \tag{13}$$

The $p(\mathbf{x}; \boldsymbol{\mu}_k, \Sigma_k)$ part of the formula can also be viewed as $p(x|z = k)$. Therefore, we in fact want to calculate the sum over a joint distribution $p(x, z) = p(z)p(x|z)$. That is, the GMM computes the marginal:

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(x|z = k) \tag{14}$$

This interpretation is useful when we want to sample from $p_{\boldsymbol{\theta}}$. It is done as follows. We can easily sample from the joint p(x,z) by firstly sampling z from p(z). The p(z) is a categorical distribution with p(z = k) = w_K for all k from $\{1, \ldots, K\}$. Afterwards, we use the z in order to sample x from $p(x|z)$. This way we will be able to select the Gaussian component and sample the x from it. After we obtain the concrete (x,z) sample, we can throw away the z since it is not needed.

2 Bonus: Maximum Likelihood Estimation [5* points]

Assume we are given a dataset $\mathcal{D} = (\mathcal{X}, \mathcal{Y})$ with $\mathcal{X} = \{(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)})\}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^D$, and $\mathcal{Y} = \{y^{(1)}, \ldots, y^{(N)}\}$ with $y^{(i)} \in \mathbb{R}$ for all $i \in \{1, \ldots, N\}$. We want to solve a supervised regression problem using a model $f_{\boldsymbol{\theta}}$ that tries to predict y from $\mathbf{x}$. For example, $f_{\boldsymbol{\theta}}$ could represent a neural network.

We will now adopt a probabilistic perspective on this problem and show the equivalence between *Maximum Likelihood Estimation* and minimizing popular loss functions for training regression models.

2.1 Tasks:

Assume that the likelihood of a point $y^{(i)}$ given the features $\mathbf{x}^{(i)}$ is Gaussian with mean $f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$ and variance σ^2 (which does not depend on $\mathbf{x}^{(i)}$), i.e.,

$$p_{\boldsymbol{\theta}}(y^{(i)} \,|\, \mathbf{x}^{(i)}) = \mathcal{N}(y^{(i)}; f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), \sigma^2) \quad \text{where} \quad \mathcal{N}(y^{(i)}; f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2\right)$$

If we assume that all $y^{(i)} \in \mathcal{Y}$ were independently drawn from their corresponding conditional Gaussian distribution, we can factorize the likelihood of $\mathcal{Y}$ as follows:

$$p_{\boldsymbol{\theta}}(\mathcal{Y} \mid \mathcal{X}) = \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(y^{(i)} \,|\, \mathbf{x}^{(i)})$$

Recall that we have often tried to find parameters of $f_{\boldsymbol{\theta}}$ by minimizing the mean squared error:

$$\boldsymbol{\theta}^*_{\text{MSE}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \left(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}\right)^2$$

Show that under the assumptions above, we have $\boldsymbol{\theta}^*_{\text{MSE}} = \boldsymbol{\theta}^*_{\text{MLE}}$ where $\boldsymbol{\theta}^*_{\text{MLE}}$ is the *maximum likelihood estimate* given by

$$\boldsymbol{\theta}^*_{\text{MLE}} = \operatorname*{argmax}_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathcal{Y} \mid \mathcal{X}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \log\left(p_{\boldsymbol{\theta}}(\mathcal{Y} \mid \mathcal{X})\right)$$

Provide all steps of your derivation.

Hints: Use the following generic facts about minimizers/maximizers:

- $\operatorname{argmin}_{\mathbf{z}} g(\mathbf{z}) = \operatorname{argmin}_{\mathbf{z}} \log(g(\mathbf{z}))$
- $\operatorname{argmax}_{\mathbf{z}} g(\mathbf{z}) = \operatorname{argmin}_{\mathbf{z}} -g(\mathbf{z})$
- $\operatorname{argmin}_{\mathbf{z}} \alpha g(\mathbf{z}) + \beta = \operatorname{argmin}_{\mathbf{z}} g(\mathbf{z})$

where g is some function and α, β are constants w.r.t. $\mathbf{z}$. Also, recall that $\forall a, b \in \mathbb{R} : \log(a \cdot b) = \log(a) + \log(b)$.

To demonstrate that minimizing the Mean Squared Error (MSE) is equivalent to maximizing the likelihood of the data assuming a Gaussian distribution, we follow these steps:

Step 1: Define the Likelihood We start by considering the likelihood of each response $y^{(i)}$ given the corresponding feature vector $\mathbf{x}^{(i)}$. It is assumed that this follows a Gaussian distribution with mean $f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$ and variance σ^2:

$$p_{\boldsymbol{\theta}}(y^{(i)} \mid \mathbf{x}^{(i)}) = \mathcal{N}(y^{(i)}; f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), \sigma^2)$$

Since the data points are independent, the joint likelihood of all data points can be expressed as the product of the individual likelihoods:

$$p_{\boldsymbol{\theta}}(\mathcal{Y} \mid \mathcal{X}) = \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(y^{(i)} \mid \mathbf{x}^{(i)})$$

Step 2: Compute the Log-Likelihood To simplify our calculations, we take the natural logarithm of the likelihood (log-likelihood), which converts the product into a sum:

$$\log(p_{\boldsymbol{\theta}}(\mathcal{Y} \mid \mathcal{X})) = \sum_{i=1}^{N} \log(p_{\boldsymbol{\theta}}(y^{(i)} \mid \mathbf{x}^{(i)}))$$

Step 3: Substitute the Gaussian Density We substitute the Gaussian probability density function into our log-likelihood expression. The Gaussian PDF is given by:

$$p_{\boldsymbol{\theta}}(y^{(i)} \mid \mathbf{x}^{(i)}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2\right)$$

Taking the logarithm of this expression, we obtain:

$$\log(p_{\boldsymbol{\theta}}(y^{(i)} \mid \mathbf{x}^{(i)})) = -\log(\sigma\sqrt{2\pi}) - \frac{1}{2\sigma^2}(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2$$

Step 4: Sum the Log-Likelihoods Summing the log-likelihoods over all data points, we get the total log-likelihood:

$$\log(p_{\boldsymbol{\theta}}(\mathcal{Y} \mid \mathcal{X})) = \sum_{i=1}^{N} \left(-\log(\sigma\sqrt{2\pi}) - \frac{1}{2\sigma^2}(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2\right)$$

Step 5: Simplify the Expression The term $-\log(\sigma\sqrt{2\pi})$ is constant with respect to $\boldsymbol{\theta}$, so it can be factored out:

$$\log(p_{\boldsymbol{\theta}}(\mathcal{Y} \mid \mathcal{X})) = -N\log(\sigma\sqrt{2\pi}) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2$$

Step 6: Maximize the Log-Likelihood To find the maximum likelihood estimate (MLE), we maximize the log-likelihood. The constant term $-N\log(\sigma\sqrt{2\pi})$ does not affect the optimization process, so we focus on the remaining part:

$$\boldsymbol{\theta}^*_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\text{argmax}} \left(-\frac{1}{2\sigma^2}\sum_{i=1}^{N}(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2 \right)$$

Step 7: Convert to Minimization Maximizing the log-likelihood is equivalent to minimizing its negative. Therefore:

$$\boldsymbol{\theta}^*_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\text{argmin}} \left(\frac{1}{2\sigma^2}\sum_{i=1}^{N}(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2 \right)$$

Step 8: Further Simplification The factor $\frac{1}{2\sigma^2}$ is constant and does not affect the location of the minimum. Hence, it can be omitted:

$$\boldsymbol{\theta}^*_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\text{argmin}} \sum_{i=1}^{N}(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2$$

Step 9: Compare with MSE The expression obtained is exactly the Mean Squared Error (MSE) objective, but without the $\frac{1}{N}$ scaling factor, which also does not affect the optimization:

$$\boldsymbol{\theta}^*_{\text{MSE}} = \underset{\boldsymbol{\theta}}{\text{argmin}} \frac{1}{N}\sum_{i=1}^{N}\left(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}\right)^2$$

Conclusion Therefore, it has been shown that:

$$\boldsymbol{\theta}^*_{\text{MSE}} = \boldsymbol{\theta}^*_{\text{MLE}}$$

In simple terms, minimizing the MSE is equivalent to maximizing the likelihood of the data assuming it follows a Gaussian distribution with constant variance.