Assignment 2

Statistical Learning Theory and Data Science
Course Code: BI3424/DS3214

The assignment submission deadline is April 20, 2025, at 11:59 PM.

Obtaining a diabetes threshold from glucose data

Glucose homeostasis refers to the body processes that strive to maintain blood glucose *in a tight range*. Type 2 diabetes (T2DM) is a disease characterized by a persistent *hyperglycemia*: Clinically, a fasting blood glucose level of 126 mg/dL or more is diagnosed as diabetes.

The goal of this exercise is to ask: If we collect data on fasting glucose from a population – some of these will have healthy glucose homeostasis and others will have elevated blood glucose – could we **compute the correct glucose threshold for defining diabetes**. That is, can we *check* whether the diagnosis threshold of 126 mg/dL is indeed appropriate?

We will use some *simulated* data for our experiment. Create a dataset containing glucose values that follow two distinct distributions:

1. **Normal glucose range:** Gaussian distribution with a mean of 80 mg/dL and standard deviation of 20 mg/dL,
2. **Hyperglycemic range:** Gaussian distribution with a mean of 220 mg/dL and standard deviation of 50 mg/dL.

Our goal is to **determine an optimal threshold value that can classify patients as diabetic or non-diabetic from this data**. We will employ three clustering algorithms – specifically K-means clustering, DBSCAN and Gaussian Mixture Models (GMM). We will then examine various sample sizes ($n = 20, 50$ and 100) to check whether that makes any difference?

Your assignment requires you to:

1. Evaluate which algorithm performs best for this classification challenge, considering that the clinical threshold is 126 mg/dL.
2. Provide a detailed explanation of why certain algorithms succeed while others fail. Does the number of data points used for clustering affect the analysis?
3. Provide clear and comprehensive visualizations of the results obtained. Your graphical representations should include:
    - The original distribution of the synthetic glucose data,
    - The resulting distributions after applying each clustering algorithm, and
    - A scatter plot displaying the identified clusters along with the calculated decision threshold.

For K-means and DBSCAN, you may utilize the built-in functions from `sklearn.cluster`. However, for GMM, you must implement the corresponding Expectation-Maximization (EM) algorithm from scratch as outlined below.

Steps to implement Gaussian Mixture Models (GMM)

1. Compute Gaussian Probability Density Function (PDF):

   Create a function `gaussian_pdf(x, mean, var)` that computes the value of the Gaussian (Normal) distribution at x for given `mean` and `variance`:

   $$\text{PDF}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

   This is used in both the E-step and for computing final "responsibilities" (see below).

2. Initialization:

   Parameters are initialized as follows: - Means (μ_k): randomly chosen from the data. - Variances (σ_k^2): initialized to the variance of the entire dataset. - Weights (w_k): initialized equally for two components, i.e., 0.5.

3. E–step (Expectation) of the E–M algorithm:

   This step computes the "responsibility" r_{nk}: the probability that the n^{th} data point was generated by the k^{th} Gaussian component.

   $$r_{nk} = \frac{w_k \cdot \mathcal{N}(x_n)}{\sum w_j \cdot \mathcal{N}(x_n)}$$

   This gives a matrix of shape `(n_samples, 2)` in our case.

4. M–step (Expectation) of the E–M algorithm:

   Update the parameters using the responsibilities from the E-step:

   - Effective number of samples assigned to each component:

   $$N_k = \sum_{n=1}^{N} r_{nk}$$

   - Updated means:

   $$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} x_n$$

   - Updated variances:

   $$\sigma_k^2 = \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} (x_n - \mu_k)^2$$

- Updated weights:

$$w_k = \frac{N_k}{N}$$

5. Final Assignment of Labels:

   After the EM loop, the algorithm re-computes the responsibilities and assigns each data point to the component with maximum responsibility (i.e., class with highest posterior probability):

$$\text{label}(x_n) = \arg\max_k \ r_{nk}$$

6. Finding the Intersection (Threshold):

   The threshold between the two Gaussians is computed by solving:

$$w_1 \cdot \mathcal{N}(x|\mu_1, \sigma_1^2) = w_2 \cdot \mathcal{N}(x|\mu_2, \sigma_2^2)$$

   This is done numerically using `fsolve`. The point where these two weighted distributions intersect can serve as a decision boundary or threshold between the two classes.

Step	**Description**	**Mathematical Expression**
1	PDF Computation	$\mathcal{N}(x)$
2	Initialize Parameters	Random for means, equal weights
3	E-step	$r_{nk} = \frac{w_k \cdot \mathcal{N}(x_n)}{\sum_j w_j \cdot \mathcal{N}(x_n)}$
4	M-step	Update μ_k, σ_k^2, w_k
5	Label Assignment	$\arg\max_k \ r_{nk}$
6	Threshold (Intersection)	$w_1 \cdot \mathcal{N}_1(x) = w_2 \cdot \mathcal{N}_2(x)$

Notes

1. Take `max_iter=100`, `tol=1e-6`.
2. Use this to find the cluster labels (see step 5):

```
labels = np.argmax(final_responsibility, axis=1)
```

3. Use this function to find the threshold:

```
def find_intersection(mu1, sigma1, w1, mu2, sigma2, w2):
def equation(x):
return w1 * norm.pdf(x, mu1, sigma1) - w2 * norm.pdf(x, mu2, sigma2)
x0 = (mu1 + mu2) / 2
return fsolve(equation, x0)[0]
```

4. To use `fsolve`, you will require `from scipy.optimize import fsolve`.