

# ML Assignment 5

Bohan YANG  
Student ID: 2330016056

December 13, 2025

## Question 1

### Data

$$X = \{(0, 2), (0, 0), (1, 0), (5, 0), (5, 2)\}.$$

### Objective

Let  $c_i \in \{1, 2\}$  be the cluster assignment of  $x_i$  and  $\mu_k$  be the centroid of cluster  $k$ . K-means minimizes the within-cluster sum of squares (WCSS):

$$J(\{\mu_k\}, \{c_i\}) = \sum_{i=1}^N \|x_i - \mu_{c_i}\|^2.$$

### Algorithm

**Assignment step.** For fixed centroids  $\{\mu_k\}$ , each point is assigned to the nearest centroid:

$$c_i = \arg \min_{k \in \{1, 2\}} \|x_i - \mu_k\|^2.$$

**Update step.** For fixed assignments, minimize

$$J_k(\mu_k) = \sum_{i: c_i = k} \|x_i - \mu_k\|^2.$$

Differentiate and set to zero:

$$\frac{\partial J_k}{\partial \mu_k} = 2 \sum_{i: c_i = k} (\mu_k - x_i) = 0 \quad \Rightarrow \quad \mu_k = \frac{1}{|S_k|} \sum_{i \in S_k} x_i,$$

where  $S_k = \{i : c_i = k\}$ .

## Results

$$S_1 = \{(0, 2), (0, 0), (1, 0)\}, \quad S_2 = \{(5, 0), (5, 2)\}.$$
$$\mu_1 = \frac{(0, 2) + (0, 0) + (1, 0)}{3} = \left(\frac{1}{3}, \frac{2}{3}\right), \quad \mu_2 = \frac{(5, 0) + (5, 2)}{2} = (5, 1).$$

## Question 2

### Model

For each trial  $i$ :

- Toss coin  $A$ :  $P(A = H) = \pi$ ,  $P(A = T) = 1 - \pi$ .
- If  $A = H$  choose coin  $B$ , else choose coin  $C$ .
- Toss the chosen coin and observe  $x_i \in \{0, 1\}$  ( $H=1$ ,  $T=0$ ).

Define latent variable  $z_i \in \{0, 1\}$ :

$$z_i = 1 \Rightarrow \text{coin } B \text{ used}, \quad z_i = 0 \Rightarrow \text{coin } C \text{ used}.$$

Parameters:

$$P(z_i = 1) = \pi, \quad P(x_i = 1 \mid z_i = 1) = p, \quad P(x_i = 1 \mid z_i = 0) = q.$$

Observed sequence ( $n = 10$ ):

$$x = (1, 1, 0, 1, 0, 0, 1, 0, 1, 1).$$

### Incomplete-data likelihood

$$P(x_i = 1) = \pi p + (1 - \pi)q, \quad P(x_i = 0) = \pi(1 - p) + (1 - \pi)(1 - q).$$

Thus

$$L(\pi, p, q) = \prod_{i=1}^n \left[ \pi p^{x_i} (1 - p)^{1-x_i} + (1 - \pi) q^{x_i} (1 - q)^{1-x_i} \right].$$

### Complete-data log-likelihood

$$\log P(X, Z) = \sum_{i=1}^n \left( z_i [\log \pi + x_i \log p + (1 - x_i) \log(1 - p)] + (1 - z_i) [\log(1 - \pi) + x_i \log q + (1 - x_i) \log(1 - q)] \right).$$

### EM derivation

#### E-step

$$\gamma_i := P(z_i = 1 \mid x_i, \theta^{(t)}) = \frac{\pi^{(t)} p^{(t) x_i} (1 - p^{(t)})^{1-x_i}}{\pi^{(t)} p^{(t) x_i} (1 - p^{(t)})^{1-x_i} + (1 - \pi^{(t)}) q^{(t) x_i} (1 - q^{(t)})^{1-x_i}}.$$

**M-step** Using  $\mathbb{E}[z_i] = \gamma_i$ :

$$\pi^{(t+1)} = \frac{1}{n} \sum_{i=1}^n \gamma_i, \quad p^{(t+1)} = \frac{\sum_{i=1}^n \gamma_i x_i}{\sum_{i=1}^n \gamma_i}, \quad q^{(t+1)} = \frac{\sum_{i=1}^n (1 - \gamma_i) x_i}{\sum_{i=1}^n (1 - \gamma_i)}.$$

## Results

Initialize:

$$\pi^{(0)} = 0.46, \quad p^{(0)} = 0.55, \quad q^{(0)} = 0.67.$$

E-step responsibilities for each observation (rounded to 6 decimals):

$$\gamma \approx (0.411516, 0.411516, 0.537383, 0.411516, 0.537383, 0.537383, 0.411516, 0.537383, 0.411516, 0.411516).$$

M-step yields (and converges immediately for this dataset):

$\hat{\pi} = 0.4618628351, \quad \hat{p} = 0.5345950038, \quad \hat{q} = 0.6561346418.$

## Question 3

### Data

$$x = (-67, -48, 6, 8, 14, 16, 23, 24, 28, 29, 41, 49, 56, 60, 75).$$

### Model

A 2-component 1D GMM:

$$p(x) = \alpha_1 \mathcal{N}(x \mid \mu_1, \sigma_1^2) + \alpha_2 \mathcal{N}(x \mid \mu_2, \sigma_2^2), \quad \alpha_1 + \alpha_2 = 1, \quad \alpha_k \geq 0.$$

$$\mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right).$$

### EM derivation

Introduce latent indicator  $z_i \in \{1, 2\}$ . Define responsibility:

$$\gamma_{ik} = P(z_i = k \mid x_i, \theta^{(t)}) = \frac{\alpha_k^{(t)} \mathcal{N}(x_i \mid \mu_k^{(t)}, \sigma_k^{2(t)})}{\sum_{j=1}^2 \alpha_j^{(t)} \mathcal{N}(x_i \mid \mu_j^{(t)}, \sigma_j^{2(t)})}.$$

Let  $N_k = \sum_{i=1}^n \gamma_{ik}$ . M-step updates:

$$\alpha_k^{(t+1)} = \frac{N_k}{n}, \quad \mu_k^{(t+1)} = \frac{1}{N_k} \sum_{i=1}^n \gamma_{ik} x_i,$$

$$\sigma_k^{2(t+1)} = \frac{1}{N_k} \sum_{i=1}^n \gamma_{ik} (x_i - \mu_k^{(t+1)})^2, \quad \sigma_k^{(t+1)} = \sqrt{\sigma_k^{2(t+1)}}.$$

## Results

$\alpha_1 = 0.8668277165,$	$\mu_1 = 32.9848875405,$	$\sigma_1 = 20.7233763208;$
$\alpha_2 = 0.1331722835,$	$\mu_2 = -57.5110766992,$	$\sigma_2 = 9.4999935621.$

## Question 4

### Data matrix

$$X = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 5 & 6 & 7 \\ 1 & 4 & 2 & 3 \\ 5 & 3 & 2 & 1 \\ 8 & 1 & 2 & 2 \end{pmatrix} \in \mathbb{R}^{5 \times 4}.$$

### Derivation and procedure

(1) **Centering.** Let column mean be  $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ . Centered data:

$$X_c = X - 1\bar{x}^\top.$$

(2) **Covariance matrix.**

$$S = \frac{1}{n-1} X_c^\top X_c \in \mathbb{R}^{4 \times 4}.$$

(3) **Eigen-decomposition.** Solve

$$Sv_j = \lambda_j v_j, \quad \lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_4.$$

Take the top two eigenvectors:

$$W = [v_1 \ v_2] \in \mathbb{R}^{4 \times 2}.$$

(4) **Projection to 2D.**

$$Z = X_c W \in \mathbb{R}^{5 \times 2}.$$

## Results

Top eigenvalues:

$$\lambda_1 = 10.6066305, \quad \lambda_2 = 7.90808697.$$

Corresponding eigenvectors:

$$W = \begin{pmatrix} -0.69478464 & 0.69892736 \\ 0.34820806 & 0.17035429 \\ 0.32341225 & 0.47997101 \\ 0.53984254 & 0.50210337 \end{pmatrix}.$$

2D embeddings  $Z = X_c W$ :

$$Z \approx \begin{pmatrix} 2.06005139 & -1.96587434 \\ 2.91530135 & 4.2871211 \\ 1.89321273 & -2.60724014 \\ -2.31381898 & -0.98609174 \\ -4.55474649 & 1.27208513 \end{pmatrix}.$$

Explained variance ratio of the first two PCs:

$$\frac{\lambda_1 + \lambda_2}{\sum_{j=1}^4 \lambda_j} \approx 0.9350867.$$

# A5

December 14, 2025

```
[ ]: import numpy as np

X = np.array([[0, 2], [0, 0], [1, 0], [5, 0], [5, 2]], dtype=float)

def kmeans(X, k=2, n_init=50, max_iter=100, seed=42):
    rng = np.random.default_rng(seed)
    n, d = X.shape

    best_inertia = np.inf
    best_centroids = None
    best_labels = None

    for _ in range(n_init):
        centroids = X[rng.choice(n, size=k, replace=False)].copy()
        labels = None

        for _ in range(max_iter):
            dist2 = ((X[:, None, :] - centroids[None, :, :]) ** 2).sum(axis=2)
            new_labels = dist2.argmin(axis=1)

            new_centroids = centroids.copy()
            for j in range(k):
                pts = X[new_labels == j]
                if len(pts) > 0:
                    new_centroids[j] = pts.mean(axis=0)
                else:
                    new_centroids[j] = X[rng.integers(n)]

            labels = new_labels
            centroids = new_centroids

        inertia = ((X - centroids[labels]) ** 2).sum()
        if inertia < best_inertia:
            best_inertia = inertia
            best_centroids = centroids.copy()
            best_labels = labels.copy()
```

```

    return best_centroids, best_labels, best_inertia

C, y, sse = kmeans(X, k=2, n_init=50, seed=42)

print("Centroids:\n", C)
print("Labels (per sample):", y)    # y[i] is the cluster id of X[i]
print("SSE (inertia):", sse)

for j in range(2):
    idx = np.where(y == j)[0]
    print(f"\nCluster {j}: indices {idx.tolist()}, points:\n{X[idx]}")

```

```

Centroids:
[[0.33333333 0.66666667]
 [5.         1.         ]]
Labels (per sample): [0 0 0 1 1]
SSE (inertia): 5.3333333333333334

Cluster 0: indices [0, 1, 2], points:
[[0. 2.]
 [0. 0.]
 [1. 0.]]

Cluster 1: indices [3, 4], points:
[[5. 0.]
 [5. 2.]]

```

```

[3]: import numpy as np

x = np.array([1, 1, 0, 1, 0, 0, 1, 0, 1, 1], dtype=float)
pi, p, q = 0.46, 0.55, 0.67

def em_three_coins(x, pi, p, q, tol=1e-12, maxit=1000):
    ll_prev = None
    for t in range(maxit):
        num = pi*(p**x)*((1-p)**(1-x))
        den = num + (1-pi)*(q**x)*((1-q)**(1-x))
        gamma = num/den

        pi_new = gamma.mean()
        p_new = (gamma*x).sum()/gamma.sum()
        q_new = ((1-gamma)*x).sum()/(1-gamma).sum()

        ll = np.sum(np.log(

```

```

        pi_new*(p_new**x)*((1-p_new)**(1-x)) +
        (1-pi_new)*(q_new**x)*((1-q_new)**(1-x))
    ))
    if ll_prev is not None and abs(ll-ll_prev) < tol:
        return pi_new, p_new, q_new, t+1, ll, gamma
    pi, p, q = pi_new, p_new, q_new
    ll_prev = ll
    return pi, p, q, maxit, ll_prev, gamma

pi_hat, p_hat, q_hat, iters, ll, gamma = em_three_coins(x, pi, p, q)
print("iters:", iters)
print("pi_hat, p_hat, q_hat:", pi_hat, p_hat, q_hat)
print("gamma:", gamma)

```

```

iters: 2
pi_hat, p_hat, q_hat: 0.46186283511391907 0.5345950037850111 0.6561346417857326
gamma: [0.41151594 0.41151594 0.53738318 0.41151594 0.53738318 0.53738318
0.41151594 0.53738318 0.41151594 0.41151594]

```

```

[ ]: import numpy as np

x = np.array([-67, -48, 6, 8, 14, 16, 23, 24, 28,
              29, 41, 49, 56, 60, 75], dtype=float)
n = len(x)

def normpdf(x, mu, s):
    return (1.0/(np.sqrt(2*np.pi)*s))*np.exp(-(x-mu)**2/(2*s**2))

def em_gmm2(x, init, tol=1e-8, maxit=1000):
    a1, mu1, s1, a2, mu2, s2 = init
    ll_prev = None
    for t in range(maxit):
        p1 = a1 * normpdf(x, mu1, s1)
        p2 = a2 * normpdf(x, mu2, s2)
        gamma1 = p1 / (p1 + p2)
        gamma2 = 1.0 - gamma1

        N1, N2 = gamma1.sum(), gamma2.sum()
        a1_new, a2_new = N1/n, N2/n
        mu1_new = (gamma1*x).sum() / N1
        mu2_new = (gamma2*x).sum() / N2
        s1_new = np.sqrt((gamma1*(x-mu1_new)**2).sum() / N1)
        s2_new = np.sqrt((gamma2*(x-mu2_new)**2).sum() / N2)

```

```

        p1n = a1_new * normpdf(x, mu1_new, s1_new)
        p2n = a2_new * normpdf(x, mu2_new, s2_new)
        ll = np.sum(np.log(p1n + p2n))

        a1, mu1, s1, a2, mu2, s2 = a1_new, mu1_new, s1_new, a2_new, mu2_new, s2_new
        ll_prev = ll

        return a1, mu1, s1, a2, mu2, s2, maxit, ll_prev

rng = np.random.default_rng(0)
best = None
for _ in range(50):
    mu1, mu2 = rng.choice(x, 2, replace=False)
    s = np.std(x, ddof=0)
    init = (0.5, mu1, s, 0.5, mu2, s)
    res = em_gmm2(x, init)
    if best is None or res[-1] > best[-1]:
        best = res

print("best:", best)

```

```

best: (np.float64(0.8668277187941408), np.float64(32.98488732303651),
np.float64(20.72337672109887), np.float64(0.13317228120585922),
np.float64(-57.51107685642921), np.float64(9.499993561952834), 1000,
np.float64(-71.06336186471681))

```

```

[ ]: import numpy as np

X = np.array([
    [1, 2, 3, 4],
    [5, 5, 6, 7],
    [1, 4, 2, 3],
    [5, 3, 2, 1],
    [8, 1, 2, 2]
], dtype=float)

n = X.shape[0]
mean = X.mean(axis=0)
Xc = X - mean
S = (Xc.T @ Xc) / (n-1)

eigvals, eigvecs = np.linalg.eigh(S)
idx = np.argsort(eigvals)[::-1]
eigvals = eigvals[idx]
eigvecs = eigvecs[:, idx]

```

```

W = eigvecs[:, :2]
Z = Xc @ W
evr = eigvals[:2].sum() / eigvals.sum()

print("eigvals:", eigvals)
print("W:\n", W)
print("Z:\n", Z)
print("Explained variance ratio:", evr)

```

```
eigvals: [10.6066305  7.90808697  1.19062586  0.09465667]
```

```
W:
```

```

[[-0.69478464  0.69892736]
 [ 0.34820806  0.17035429]
 [ 0.32341225  0.47997101]
 [ 0.53984254  0.50210337]]

```

```
Z:
```

```

[[ 2.06005139 -1.96587434]
 [ 2.91530135  4.2871211 ]
 [ 1.89321273 -2.60724014]
 [-2.31381898 -0.98609174]
 [-4.55474649  1.27208513]]

```

```
Explained variance ratio: 0.9350867411323983
```

```
[ ]:
```