A5

December 14, 2025

[ ]:
```python
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): 5.333333333333334

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]
```

```python
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
```

```
[ ]:
```