Homework 3

Amir Ebrahimi

1. $P(x\,|\,\omega_2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$ and $P(x\,|\,\omega_2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-1)^2}{2\sigma^2}\right)$. If we write the Likelihood ratio test for the two-class classification problem:

$\dfrac{P(x\,|\,\omega_1)}{P(x\,|\,\omega_2)} \lessgtr \dfrac{\lambda_{21}P(\omega_2)}{\lambda_{12}P(\omega_1)}$, we can substitute the Gaussian PDFs and get:

$$\exp\left(-\frac{(x-1)^2 - x^2}{2\sigma^2}\right) \lessgtr \frac{\lambda_{21}P(\omega_2)}{\lambda_{12}P(\omega_1)} \implies \exp\left(-\frac{x^2 - 2x + 1 - x^2}{2\sigma^2}\right) \lessgtr \frac{\lambda_{21}P(\omega_2)}{\lambda_{12}P(\omega_1)}$$

$\implies \exp\left(-\dfrac{-2x+1}{2\sigma^2}\right) \lessgtr \dfrac{\lambda_{21}P(\omega_2)}{\lambda_{12}P(\omega_1)}$. Taking $\ln(\,.\,)$:

$-\dfrac{-2x+1}{2\sigma^2} \lessgtr \ln\left(\dfrac{\lambda_{21}P(\omega_2)}{\lambda_{12}P(\omega_1)}\right)$. Solving for x: $\dfrac{2x-1}{2\sigma^2} \lessgtr \ln\left(\dfrac{\lambda_{21}P(\omega_2)}{\lambda_{12}P(\omega_1)}\right) \implies$

$2x \lessgtr 1 + 2\sigma^2 \ln\left(\dfrac{\lambda_{21}P(\omega_2)}{\lambda_{12}P(\omega_1)}\right) \implies x \lessgtr \dfrac{1}{2} + \sigma^2 \ln\left(\dfrac{\lambda_{21}P(\omega_2)}{\lambda_{12}P(\omega_1)}\right)$. Thus, the threshold for

Average risk minimization is: $x_0 = \dfrac{1}{2} - \sigma^2 \ln\left(\dfrac{\lambda_{21}P(\omega_2)}{\lambda_{12}P(\omega_1)}\right)$.

2. Summing over all samples:

$$\sum_{k=1}^{N} (\mathbf{x}_k - \hat{\mu})(\mathbf{x}_k - \hat{\mu})^T = \sum_{k=1}^{N} \mathbf{x}_k \mathbf{x}_k^T - \sum_{k=1}^{N} \mathbf{x}_k \hat{\mu}^T - \sum_{k=1}^{N} \hat{\mu}\,\mathbf{x}_k^T + \sum_{k=1}^{N} \hat{\mu}\hat{\mu}^T \implies$$

$\displaystyle\sum_{k=1}^{N} \mathbf{x}_k \hat{\mu}^T = \sum_{k=1}^{N} \mathbf{x}_k \left(\frac{1}{N}\sum_{j=1}^{N} \mathbf{x}_j\right)^T = \frac{1}{N}\sum_{k=1}^{N}\sum_{j=1}^{N} \mathbf{x}_k \mathbf{x}_j^T$ and $\displaystyle\sum_{k=1}^{N} \hat{\mu}\,\mathbf{x}_k^T = \frac{1}{N}\sum_{k=1}^{N}\sum_{j=1}^{N} \mathbf{x}_j \mathbf{x}_k^T$ and

$$\sum_{k=1}^{N} \hat{\mu}\hat{\mu}^T = N\hat{\mu}\hat{\mu}^T = N\left(\frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i\right)\left(\frac{1}{N}\sum_{j=1}^{N} \mathbf{x}_j\right)^T = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} \mathbf{x}_i \mathbf{x}_j^T \implies$$

$$\sum_{k=1}^{N} (\mathbf{x}_k - \hat{\mu})(\mathbf{x}_k - \hat{\mu})^T = \sum_{k=1}^{N} \mathbf{x}_k \mathbf{x}_k^T - 2\left(\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} \mathbf{x}_i \mathbf{x}_j^T\right) + \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} \mathbf{x}_i \mathbf{x}_j^T \;=$$

$$\sum_{k=1}^{N} \mathbf{x}_k \mathbf{x}_k^T - \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} \mathbf{x}_i \mathbf{x}_j^T.$$

The expectation of each term is:

$\mathbb{E}\left[\sum_{k=1}^{N} \mathbf{x}_k \mathbf{x}_k^T\right] = N\Sigma$ and $\mathbb{E}\left[\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} \mathbf{x}_i \mathbf{x}_j^T\right] = \frac{1}{N} \cdot N \cdot N \cdot \Sigma = N\Sigma.$

Expectation of $\hat{\Sigma}$: $\mathbb{E}[\hat{\Sigma}] = \mathbb{E}\left[\frac{1}{N-1}\left(\sum_{k=1}^{N} \mathbf{x}_k \mathbf{x}_k^T - \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} \mathbf{x}_i \mathbf{x}_j^T\right)\right]$. Substituting calculated expectations:

$\mathbb{E}[\hat{\Sigma}] = \frac{1}{N-1}(N\Sigma - N\Sigma) = \Sigma$. Therefore, $\hat{\Sigma}$ is an unbiased estimator of the true covariance matrix.

3. Likelihood function: $L(\theta, \sigma^2 \mid x_1, x_2, \ldots, x_N) = \prod_{k=1}^{N} p(x_k) =$

$L(\theta, \sigma^2 \mid x_1, x_2, \ldots, x_N) = \prod_{k=1}^{N} \frac{1}{\sigma x_k \sqrt{2\pi}} \exp\left(-\frac{(\ln x_k - \theta)^2}{2\sigma^2}\right)$. Taking $\log(\,.\,) \implies$

$$\ln L(\theta, \sigma^2 \mid x_1, x_2, \ldots, x_N) = \sum_{k=1}^{N} \ln\left(\frac{1}{\sigma x_k \sqrt{2\pi}} \exp\left(-\frac{(\ln x_k - \theta)^2}{2\sigma^2}\right)\right)$$

$$= \sum_{k=1}^{N}\left(\ln\left(\frac{1}{\sigma x_k \sqrt{2\pi}}\right) - \frac{(\ln x_k - \theta)^2}{2\sigma^2}\right) = \sum_{k=1}^{N}\left(-\ln \sigma - \ln x_k - \frac{1}{2}\ln(2\pi) - \frac{(\ln x_k - \theta)^2}{2\sigma^2}\right)$$

$$= -N \ln \sigma - \sum_{k=1}^{N} \ln x_k - \frac{N}{2}\ln(2\pi) - \frac{1}{2\sigma^2}\sum_{k=1}^{N}(\ln x_k - \theta)^2$$

Taking derivative of the log-likelihood function:

$$\frac{\partial}{\partial \theta} \ln L(\theta, \sigma^2 \mid x_1, x_2, \ldots, x_N) = \frac{\partial}{\partial \theta}\left(-\frac{1}{2\sigma^2}\sum_{k=1}^{N}(\ln x_k - \theta)^2\right) = 0 \implies$$

$$-\frac{1}{2\sigma^2}\sum_{k=1}^{N}\frac{\partial}{\partial \theta}(\ln x_k - \theta)^2 = 0 \implies \frac{1}{\sigma^2}\sum_{k=1}^{N}(\ln x_k - \theta) = 0 \implies \sum_{k=1}^{N}(\ln x_k - \theta) = 0 \implies$$

$\sum_{k=1}^{N} \ln x_k - N\theta = 0 \implies \theta = \frac{1}{N}\sum_{k=1}^{N} \ln x_k$. Therefor, the ML estimate of θ is:

$$\hat{\theta}_{ML} = \frac{1}{N}\sum_{k=1}^{N} \ln x_k.$$

4. **A)** $u = \dfrac{x - a_i}{b} \implies du = \dfrac{dx}{b} \implies dx = b\, du$

$$\int_{-\infty}^{\infty} \frac{1}{\pi b} \frac{1}{1 + \left(\frac{x - a_i}{b}\right)^2} dx = \int_{-\infty}^{\infty} \frac{1}{\pi b} \frac{1}{1 + u^2} b\, du \implies \int_{-\infty}^{\infty} \frac{1}{\pi} \frac{1}{1 + u^2} du$$

The integrated $\frac{1}{\pi}\frac{1}{1+u^2}$ is the probability density function of the standard Cauchy distribution centered at 0 with scale parameter 1. We know that:

$$\int_{-\infty}^{\infty} \frac{1}{1 + u^2} du = \pi \implies \int_{-\infty}^{\infty} \frac{1}{\pi} \frac{1}{1 + u^2} du = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{1}{1 + u^2} du = \frac{1}{\pi} \cdot \pi = 1.$$

Therefore, the given statement in the problem is integrated to 1.

B) Given $f(x \mid \omega_i) = \dfrac{1}{\pi b} \dfrac{1}{1 + \left(\frac{x - a_i}{b}\right)^2}$, we want to evaluate this at $x = \frac{a_1 + a_2}{2}$:

Substituting $x = \frac{a_1 + a_2}{2}$ into $f(x \mid \omega_1)$: $f\left(\dfrac{a_1 + a_2}{2} \mid \omega_1\right) = \dfrac{1}{\pi b} \dfrac{1}{1 + \left(\frac{\frac{a_1 + a_2}{2} - a_1}{b}\right)^2}$

$$= \frac{\frac{a_1 + a_2}{2} - a_1}{b} = \frac{a_2 - a_1}{2b} \implies f\left(\frac{a_1 + a_2}{2} \mid \omega_1\right) = \frac{1}{\pi b} \frac{1}{1 + \left(\frac{a_2 - a_1}{2b}\right)^2}$$

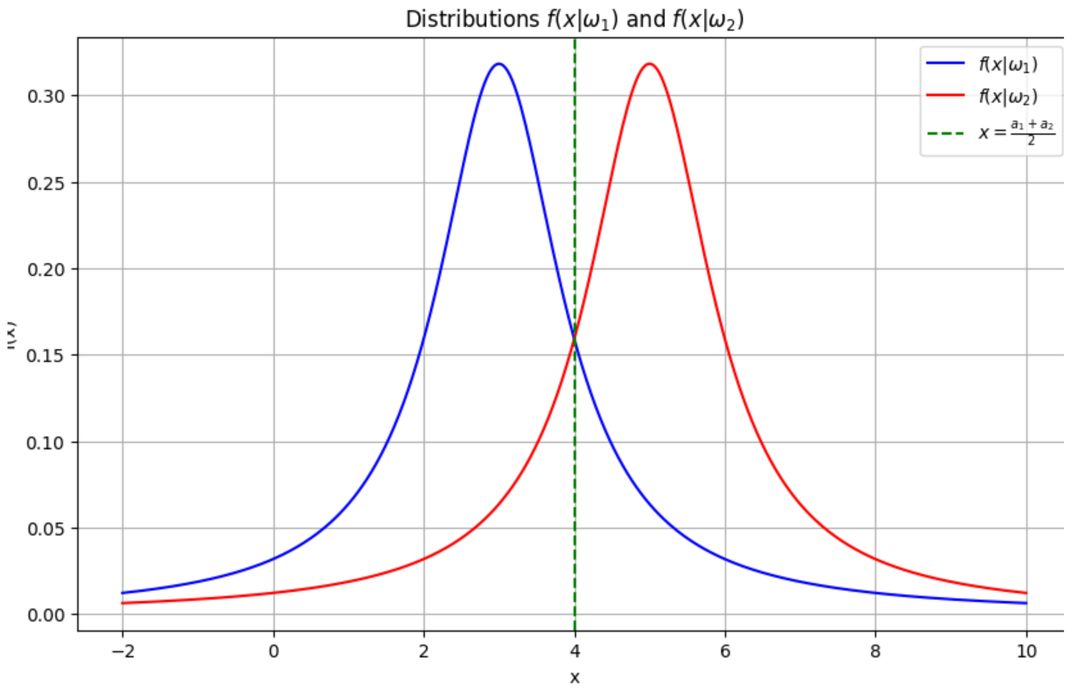
Substituting $x = \frac{a_1 + a_2}{2}$ into $f(x \mid \omega_2)$: $f\left(\dfrac{a_1 + a_2}{2} \mid \omega_2\right) = \dfrac{1}{\pi b} \dfrac{1}{1 + \left(\frac{\frac{a_1 + a_2}{2} - a_2}{b}\right)^2}$

$= \dfrac{\frac{a_1 + a_2}{2} - a_2}{b} = \dfrac{a_1 - a_2}{2b}$. Since $(a_1 - a_2)^2 = (a_2 - a_1)^2$, we have:

$f\left(\dfrac{a_1 + a_2}{2} \mid \omega_2\right) = \dfrac{1}{\pi b} \dfrac{1}{1 + \left(\frac{a_2 - a_1}{2b}\right)^2}$. And

$f\left(\dfrac{a_1 + a_2}{2} \mid \omega_1\right) = f\left(\dfrac{a_1 + a_2}{2} \mid \omega_2\right) = \dfrac{1}{\pi b} \dfrac{1}{1 + \left(\frac{a_2 - a_1}{2b}\right)^2}$. Therefore,

$$P(\omega_1 \mid x = \frac{a_1 + a_2}{2}) = P(\omega_2 \mid x = \frac{a_1 + a_2}{2}).$$



For larger $|x|$, the term $\left(\frac{x-a_i}{b}\right)^2$ dominates the constant 1 in the denominator:

$$f(x \mid \omega_i) \approx \frac{1}{\pi b} \frac{1}{\left(\frac{x-a_i}{b}\right)^2} = \frac{1}{\pi b} \frac{b^2}{(x-a_i)^2} = \frac{b}{\pi (x-a_i)^2}$$

This approximation shows that the probability density function $f(x \mid \omega_i)$ behaves asymptotically as $\frac{1}{x^2}$ for large $|x|$.

C) For the given Cauchy distributions: $f(x \mid \omega_i) = \dfrac{1}{\pi b} \dfrac{1}{1+\left(\frac{x-a_i}{b}\right)^2}$

The decision boundary is determined by:

$$f(x \mid \omega_1) = f(x \mid \omega_2) \implies \left(\frac{x-a_1}{b}\right)^2 = \left(\frac{x-a_2}{b}\right)^2 \implies$$

$x - a_1 = \pm (x - a_2)$. Thus, $a_2 = a_1$ or $x = \dfrac{a_1 + a_2}{2}$ and the decision boundary is $x = \frac{a_1+a_2}{2}$. The probability of error is the area under the PDF where classification is incorrect:

$$P(\text{error}) = P(x < \frac{a_1 + a_2}{2} \mid \omega_2)P(\omega_2) + P(x > \frac{a_1 + a_2}{2} \mid \omega_1)P(\omega_1)$$

Since $P(\omega_1) = P(\omega_2) = \frac{1}{2}$:

$$P(\text{error}) = \frac{1}{2}\left[P(x < \frac{a_1 + a_2}{2} \mid \omega_2) + P(x > \frac{a_1 + a_2}{2} \mid \omega_1)\right]$$

Given the symmetry of the Cauchy distribution and the properties of the error integral, the probability of error is:

$$P(\text{error}) = \frac{1}{2} - \frac{1}{\pi} \tan^{-1}\left(\frac{|a_2 - a_1|}{2b}\right)$$

D) To find the maximum value of $P(\text{error})$, we should analyze how does this function behave:

$\frac{1}{2} - \frac{1}{\pi} \tan^{-1}\left(\frac{|a_2 - a_1|}{2b}\right)$. The arctangent function $\tan^{-1}(x)$ ranges from

$-\frac{\pi}{2}$ to $\frac{\pi}{2}$ as x ranges from $-\infty$ to ∞. Since $\frac{|a_2 - a_1|}{2b} \geq 0$, we consider $\tan^{-1}(x)$

for $x \geq 0$. When $\frac{|a_2 - a_1|}{2b} \to 0$: $\tan^{-1}\left(\frac{|a_2 - a_1|}{2b}\right) \to 0.$

When $\frac{|a_2 - a_1|}{2b} \to \infty$: $\tan^{-1}\left(\frac{|a_2 - a_1|}{2b}\right) \to \frac{\pi}{2}$

$P(\text{error}) = \frac{1}{2} - \frac{1}{\pi} \cdot \frac{\pi}{2} = \frac{1}{2} - \frac{1}{2} = 0$. Thus, the maximum value of $P(\text{error})$

is $\frac{1}{2}$ and occurs when the location parameters a_1 and a_2 are equal $a_1 = a_2$.

E) Decision rule: $\frac{f(x \mid \omega_1)}{f(x \mid \omega_2)} \gtrless 1 \Longrightarrow f(x \mid \omega_1) = f(x \mid \omega_2)$.

Substituting the Cauchy distributions, we get:

$$\frac{1}{\pi b} \frac{1}{1 + \left(\frac{x - a_1}{b}\right)^2} = \frac{1}{\pi b} \frac{1}{1 + \left(\frac{x - a_2}{b}\right)^2}$$

$$\Longrightarrow \left(\frac{x-a_1}{b}\right)^2 = \left(\frac{x-a_2}{b}\right)^2 \Longrightarrow \left|\frac{x-a_1}{b}\right| = \left|\frac{x-a_2}{b}\right| \Longrightarrow$$

$x - a_1 = x - a_2$ or $x - a_1 = -(x - a_2)$. The first solution is trivial and always true, so we use the second solution: $x - a_1 = -(x - a_2) \Longrightarrow x = \frac{a_1 + a_2}{2}$.

$P(\text{error}) = P(x < \frac{a_1 + a_2}{2} \mid \omega_2)P(\omega_2) + P(x > \frac{a_1 + a_2}{2} \mid \omega_1)P(\omega_1)$. Given that

$P(\omega_1) = P(\omega_2) = \frac{1}{2}$,

$P(\text{error}) = \frac{1}{2}\left[P(x < \frac{a_1 + a_2}{2} \mid \omega_2) + P(x > \frac{a_1 + a_2}{2} \mid \omega_1)\right] \Longrightarrow$

$P(x < \frac{a_1 + a_2}{2} \mid \omega_2) = P(x > \frac{a_1 + a_2}{2} \mid \omega_1)$. Therefore, the probability of error is :

$$P(\text{error}) = \frac{1}{2} - \frac{1}{\pi}\tan^{-1}\left(\frac{|a_2 - a_1|}{2b}\right).$$

F) Decision rule:

Expected loss for deciding ω_1:

$$R(\omega_1 \mid x) = \lambda_{11}P(\omega_1 \mid x) + \lambda_{12}P(\omega_2 \mid x) = P(\omega_2 \mid x)$$

Expected loss for deciding ω_2

$$R(\omega_2 \mid x) = \lambda_{21}P(\omega_1 \mid x) + \lambda_{22}P(\omega_2 \mid x) = 2P(\omega_1 \mid x)$$

Minimize risk by choosing ω_1 if $P(\omega_2 \mid x) < 2P(\omega_1 \mid x) \Longrightarrow$

$$\frac{f(x \mid \omega_2)}{f(x \mid \omega_1) + f(x \mid \omega_2)} < 2\frac{f(x \mid \omega_1)}{f(x \mid \omega_1) + f(x \mid \omega_2)} \Longrightarrow f(x \mid \omega_2) < 2f(x \mid \omega_1)$$

For Cauchy distributions: $\dfrac{1 + \left(\frac{x-a_1}{b}\right)^2}{1 + \left(\frac{x-a_2}{b}\right)^2} < 2 \Longrightarrow$

$$1 + \left(\frac{x - a_1}{b}\right)^2 < 2\left[1 + \left(\frac{x - a_2}{b}\right)^2\right] \implies$$

$\left(\frac{x - a_1}{b}\right)^2 - 2\left(\frac{x - a_2}{b}\right)^2 < 1$. Thus, the decision rule is:

$(x - a_1)^2 - 2(x - a_2)^2 < b^2$.

5.

Fig. (a): 4 - The decision boundary is non-linear, indicating the use of a polynomial kernel with a low C.

Fig. (b): 1 - The decision boundary is linear, suggesting a linear SVM with moderate penalty on misclassifications.

Fig. (c): 2 - The decision boundary is linear with potentially more misclassifications, indicating a lower C value.

Fig. (d): 3 - The highly flexible, non-linear boundary shows the use of an RBF kernel with a low C.

5.

```python
from scipy.stats import multivariate_normal
import numpy as np

mean = np.array([0, 1])
cov = np.array([[1, 0], [0, 1]])

x1 = np.array([0.2, 1.3])
x2 = np.array([2.2, -1.3])

p_x1 = multivariate_normal.pdf(x1, mean=mean, cov=cov)
p_x2 = multivariate_normal.pdf(x2, mean=mean, cov=cov)

print(f"At x1: {p_x1:.4f}, At x2: {p_x2:.4f}")
```

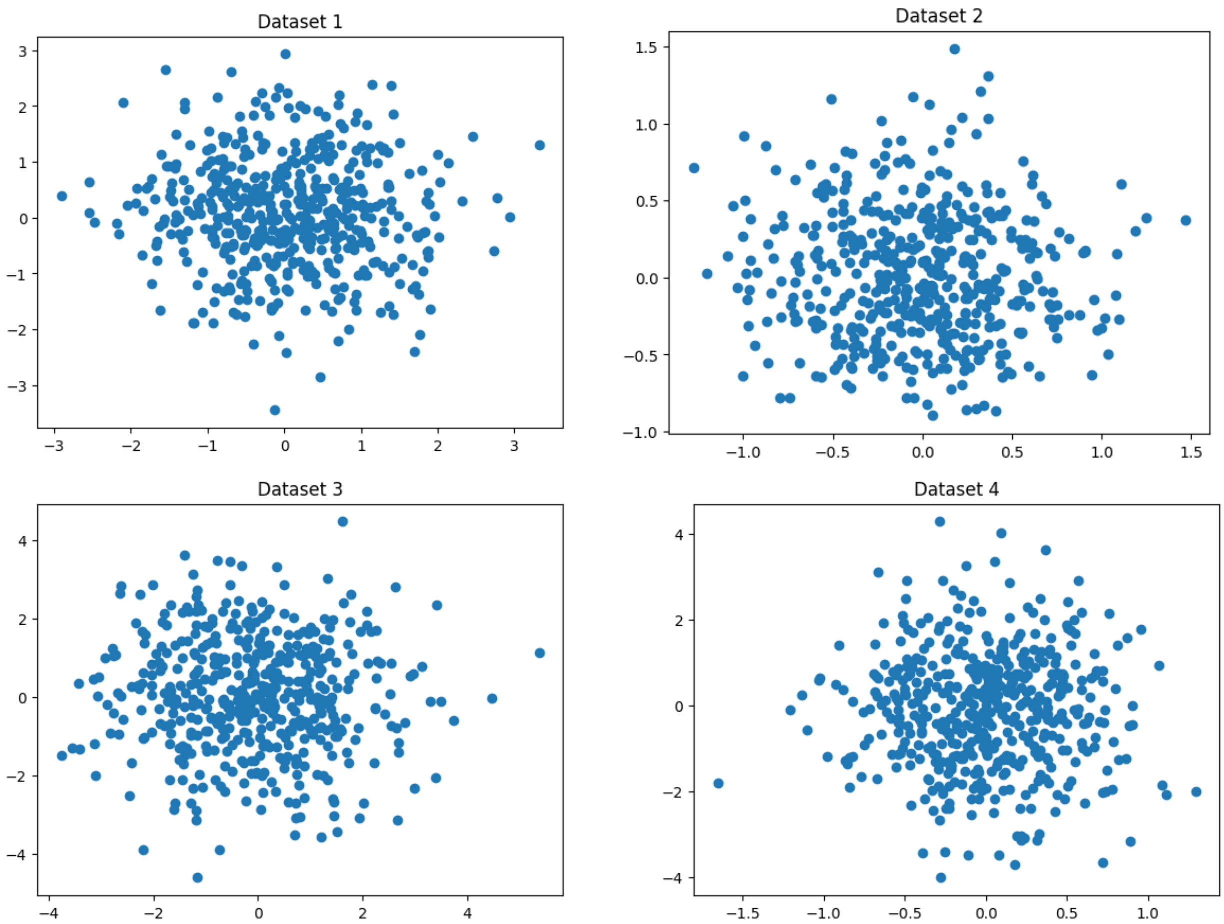
✓ 0.0s Python

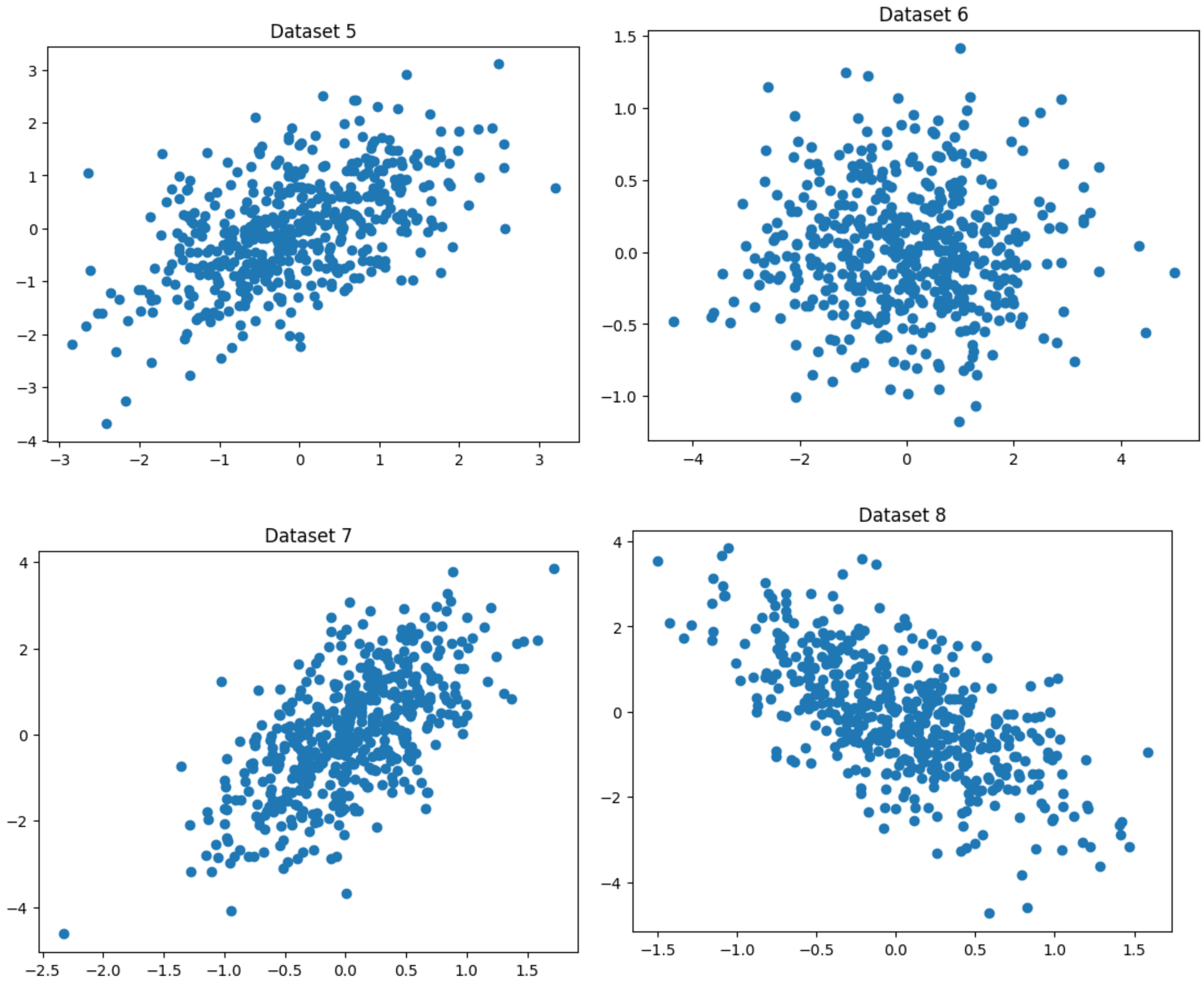
```
At x1: 0.1491, At x2: 0.0010
```

6.

In the covariance matrix, the diagonal elements shows the distribution of data points along the x and y axes. Larger values lead to a wider spread more variance), and smaller values lead to a narrower spread (less variance).

The off-diagonal elements demonstrate the covariance of two variables.
If they are positive, two variables tend to increase together leading to an ellipsoid shape along the 45-degree line in positive direction.
If one is negative, the variables' differences are in the positive direction leading to an ellipsoid shape along the 45-degree line in negative direction.
If they are zero, the variables are uncorrelated, and the distribution aligns with the axes, resulting in an axis-aligned ellipsoid or a circle if the variances are equal.





The code for generating plots:

```python
import matplotlib.pyplot as plt
```
✓ 0.2s Python

```python
rng = np.random.default_rng()
```
✓ 0.0s Python

```python
mean = ...
cov = ...

data = rng.multivariate_normal(mean, cov, 500)

plt.scatter(data[:, 0], data[:, 1])
plt.title("Dataset 8")
```
✓ 0.0s Python

7.

```python
# Gaussian 1's mean
mean1 = [1, 1]

# Gaussian 2's mean
mean2 = [3, 3]

# Covariance matrix for both
cov = [[1, 0], [0, 1]]

# Class priors
p_omega1 = p_omega2 = 0.5

# Data point
x = [1.8, 1.8]

# Gaussian 1 PDF
p_x_omega1 = multivariate_normal.pdf(x, mean=mean1, cov=cov)

# Gaussian 2 PDF
p_x_omega2 = multivariate_normal.pdf(x, mean=mean2, cov=cov)
```

```python
if p_x_omega1 > p_x_omega2:

    print("Class 1")

elif p_x_omega2 > p_x_omega1:

    print("Class 2")
```

✓ 0.0s

```
Class 1
```

8.

A) To find the mode, we need to compare $f_X(x \mid \lambda)$ and $f_X(x+1 \mid \lambda)$:

$$\frac{f_X(x+1 \mid \lambda)}{f_X(x \mid \lambda)} = \frac{\frac{e^{-\lambda}\lambda^{x+1}}{(x+1)!}}{\frac{e^{-\lambda}\lambda^{x}}{x!}} = \frac{\lambda}{x+1}$$. For the mode, we need: $\frac{f_X(x+1 \mid \lambda)}{f_X(x \mid \lambda)} \leq 1$

$\implies \frac{\lambda}{x+1} \leq 1 \implies \lambda \leq x+1$ and $x \geq \lambda - 1$. Since x is an integer, the largest integer that satisfies this inequality is $\lfloor \lambda \rfloor$.

B) $e^{-(\lambda_1-\lambda_2)}\left(\frac{\lambda_1}{\lambda_2}\right)^x > 1 \implies -(\lambda_1-\lambda_2) + x\ln\left(\frac{\lambda_1}{\lambda_2}\right) > 0 \implies$
$x\ln\left(\frac{\lambda_1}{\lambda_2}\right) > \lambda_1 - \lambda_2 \implies x > \frac{\lambda_1-\lambda_2}{\ln\left(\frac{\lambda_1}{\lambda_2}\right)}$.

9.

```python
# 9
point = np.array([0.1, 0.5, 0.1])
mean1 = np.array([0.0, 0.0, 0.0])
mean2 = np.array([0.5, 0.5, 0.5])
cov = np.array([[0.8, 0.01, 0.01],
                [0.01, 0.2, 0.01],
                [0.01, 0.01, 0.2]])
```
✓ 0.0s Python

Euclidean distance classifier:

```python
# 9
def euc_dist_classifier(point:np.ndarray, mean1:np.ndarray, mean2:np.ndarray) -> str:

    dist1 = np.linalg.norm(point - mean1)
    dist2 = np.linalg.norm(point - mean2)

    if dist1 <= dist2:
        return "Class 1"

    else:
        return "Class 2"

decision = euc_dist_classifier(point=point, mean1=mean1, mean2=mean2)
print(decision)
```
✓ 0.0s Python

```
Class 1
```

Mahalanobis distance classifier:

```python
def mah_dist_classifier(point: np.ndarray, mean1: np.ndarray,
                        mean2: np.ndarray, cov: np.ndarray) -> str:

    inv_cov = np.linalg.inv(cov)

    point_mean1_dist = point - mean1
    mah_dist1 = np.sqrt(np.dot(np.dot(point_mean1_dist.T, inv_cov), point_mean1_dist))

    point_mean2_dist = point - mean2
    mah_dist2 = np.sqrt(np.dot(np.dot(point_mean2_dist.T, inv_cov), point_mean2_dist))

    if mah_dist1 <= mah_dist2:
        return "Class 1"
    else:
        return "Class 2"

decision = mah_dist_classifier(point=point, mean1=mean1, mean2=mean2, cov=cov)
print(decision)
```

✓ 0.0s Python

```
Class 2
```

When the covariance matrix is non-symmetric (such as this case), the Mahalanobis distance classifier is more accurate. Because it considers the correlation between variables and normalizes the data based on its distribution.

10.

```python
# 10

mean = np.array([2, -2])
cov = np.array([[0.9, 0.2], [0.2, 0.3]])

np.random.seed(21)
data = np.random.multivariate_normal(mean, cov, 50)

mean_ml = np.sum(data, axis=0) / data.shape[0]

data_centered = data - mean_ml
cov_ml = np.dot(data_centered.T, data_centered) / data.shape[0]

print("ML estimate of mean: ", mean_ml)
print("ML estimate of covariance: ", cov_ml)
```

✓ 0.0s Python

```
ML estimate of mean:  [ 1.95868971 -1.9132445 ]
ML estimate of covariance:  [[0.87303966 0.11092212]
 [0.11092212 0.33511757]]
```

11.

```python
# 11

np.random.seed(21)

# Class 1
mean1 = np.array([0.0, 0.0, 0.0])

# Class 2
mean2 = np.array([1, 2, 2])

# Class 3
mean3 = np.array([3, 3, 4])

# Covariance matrix
cov = np.array([[0.8, 0.0, 0.0],
                [0.0, 0.8, 0.0],
                [0.0, 0.0, 0.8]])
```

```python
n = 1000

X_train1 = np.random.multivariate_normal(mean1, cov, n//3)
X_train2 = np.random.multivariate_normal(mean2, cov, n//3)
X_train3 = np.random.multivariate_normal(mean3, cov, n//3)

X_train = np.vstack((X_train1, X_train2, X_train3))
y_train = np.array([1]*(n//3) + [2]*(n//3) + [3]*(n//3))

X_test_1 = np.random.multivariate_normal(mean1, cov, n//3)
X_test_2 = np.random.multivariate_normal(mean2, cov, n//3)
X_test_3 = np.random.multivariate_normal(mean3, cov, n//3)

X_test = np.vstack((X_test_1, X_test_2, X_test_3))
y_test = np.array([1]*(n//3) + [2]*(n//3) + [3]*(n//3))

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```python
def initialize_parameters(X, K):
    n, d = X.shape
    np.random.seed(42)

    # Initialize parameters randomly from the data points

    mu = X[np.random.choice(n, K, replace=False)]

    cov = np.array([np.cov(X, rowvar=False) for _ in range(K)])

    pi = np.random.dirichlet(np.ones(K), size=1)[0]

    return mu, cov, pi
```

```python
def em_algorithm(X, K, max_iter=100, tol=1e-6):
    n, d = X.shape

    mu, cov, pi = initialize_parameters(X, K)

    log_likelihoods = []

    for iteration in range(max_iter):
        # E-step
        r = np.zeros((n, K))
        for k in range(K):
            r[:, k] = pi[k] * multivariate_normal.pdf(X, mean=mu[k], cov=cov[k])
        r = r / r.sum(axis=1, keepdims=True)

        # M-step
        N_k = r.sum(axis=0)
        pi = N_k / n
        mu = (r.T @ X) / N_k[:, np.newaxis]
        cov = np.zeros((K, d, d))
        for k in range(K):
            diff = X - mu[k]
            cov[k] = (r[:, k][:, np.newaxis] * diff).T @ diff / N_k[k]

        # Log-likelihood
        log_likelihood = np.sum(np.log(np.sum([pi[k] * multivariate_normal.pdf(X, mean=mu[k], cov=cov[k]) for k in range(K)], axis=0)))
        log_likelihoods.append(log_likelihood)

        # Check for convergence
        if iteration > 0 and np.abs(log_likelihood - log_likelihoods[-2]) < tol:
            print(f"Converged in {iteration} iterations")
            break

    return mu, cov, pi
```

```python
K = 3
mu, cov, pi = em_algorithm(X_train, K)

print("Estimated means:\n", mu)
print("Estimated covariance matrices:\n", cov)
print("Estimated mixing coefficients:\n", pi)
```

✓ 0.0s

```
Converged in 93 iterations
Estimated means:
 [[2.97098512e+00 2.95896480e+00 4.08558557e+00]
 [1.01215164e+00 2.09964044e+00 2.06359056e+00]
 [2.10131506e-02 4.55704299e-02 1.40876366e-03]]
Estimated covariance matrices:
 [[[ 0.85144153 -0.00263375  0.00786226]
  [-0.00263375  0.73163562  0.0639588 ]
  [ 0.00786226  0.0639588   0.863186  ]]

 [[ 0.7961972  -0.00154956 -0.0792679 ]
  [-0.00154956  0.73385162 -0.03648097]
  [-0.0792679  -0.03648097  0.65410439]]

 [[ 0.79117037  0.0483176   0.06799605]
  [ 0.0483176   0.87648905  0.08484608]
  [ 0.06799605  0.08484608  0.80452596]]]
Estimated mixing coefficients:
 [0.34349008 0.29783016 0.35867976]
```

```python
cov_avg = (cov[0] + cov[1] + cov[2]) / 3
cov_avg
```
✓ 0.0s

```
array([[ 0.81293637,  0.01471143, -0.00113653],
       [ 0.01471143,  0.78065876,  0.0374413 ],
       [-0.00113653,  0.0374413 ,  0.77393878]])
```

```python
from scipy.spatial.distance import cdist
```
✓ 0.0s

```python
# Euclidean distance classifier
def euclidean_classifier(X, means):
    distances = cdist(X, means, 'euclidean')
    return np.argmin(distances, axis=1) + 1

y_pred_euclidean = euclidean_classifier(X=X_test, means=mu)
error_prob_euclidean = np.mean(y_pred_euclidean != y_test)

print("Error probability:", error_prob_euclidean)
```
✓ 0.0s

```
Error probability: 0.7127127127127127
```

```python
# Mahalanobis distance classifier
def mahalanobis_classifier(X, means, cov_inv):
    distances = cdist(X, means, 'mahalanobis', VI=cov_inv)
    return np.argmin(distances, axis=1) + 1


cov_inv = np.linalg.inv(cov_avg)
y_pred_mahalanobis = mahalanobis_classifier(X=X_test, means=mu, cov_inv=cov_inv)
error_prob_mahalanobis = np.mean(y_pred_mahalanobis != y_test)

print("Error probability:", error_prob_mahalanobis)
```
✓ 0.0s

```
Error probability: 0.7127127127127127
```

```python
# Bayesian classifier
def bayesian_classifier(X, means, cov):
    probs = np.zeros((X.shape[0], K))
    for i, mean in enumerate(means):
        rv = multivariate_normal(mean, cov)
        probs[:, i] = rv.pdf(X)
    return np.argmax(probs, axis=1) + 1

y_pred_bayesian = bayesian_classifier(X=X_test, means=mu, cov=cov_avg)
error_prob_bayesian = np.mean(y_pred_bayesian != y_test)

print("Error probability:", error_prob_bayesian)
```
✓ 0.0s

```
Error probability: 0.7127127127127127
```

Reasons that all classifiers perform identically:

- Equal covariance assumption
- Data comes from Gaussian distribution
- Classes are equiprobable

12.

```
# 12

n = 500
means = [np.array([1, 1]), np.array([3, 3]), np.array([2, 6])]
covariances = [0.1 * np.eye(2), 0.2 * np.eye(2), 0.3 * np.eye(2)]
priors = [0.4, 0.4, 0.2]

data = np.zeros((n, 2))
labels = np.zeros(n, dtype=int)
for i in range(n):
    k = np.random.choice(len(priors), p=priors)
    data[i] = np.random.multivariate_normal(means[k], covariances[k])
    labels[i] = k
```
✓ 0.0s

```
# Initial conditions
initial_conditions = [
    (3, np.array([[0, 2], [5, 2], [5, 5]]), [0.1 * np.eye(2), 0.2 * np.eye(2), 0.3 * np.eye(2)], [1/3, 1/3, 1/3]),
    (3, np.array([[1.6, 1.4], [1.4, 1.6], [1.3, 1.5]]), [0.2 * np.eye(2), 0.4 * np.eye(2), 0.3 * np.eye(2)], [0.2, 0.4, 0.4]),
    (2, np.array([[1.6, 1.4], [1.4, 1.6]]), [0.2 * np.eye(2), 0.4 * np.eye(2)], [1/2, 1/2])
]

results = []
for (J, initial_means, initial_covariances, initial_priors) in initial_conditions:
    mu, cov, pi = em_algorithm(data, J)
    results.append((mu, cov, pi))
    print(f"Initial Means:\n{initial_means}")
    print(f"Estimated Means:\n{mu}")
    print(f"Estimated Covariances:\n{cov}")
    print(f"Estimated Priors:\n{pi}\n")
```
✓ 0.0s

```
Converged in 22 iterations
Initial Means:
[[0 2]
 [5 2]
 [5 5]]
Estimated Means:
[[1.93572557 5.987967  ]
 [1.00635204 1.02769421]
 [2.98947138 3.08185182]]
Estimated Covariances:
[[[ 0.29018311 -0.01068378]
  [-0.01068378  0.2769965 ]]

 [[ 0.10352005  0.00074768]
  [ 0.00074768  0.08863473]]

 [[ 0.19289057 -0.00139572]
  [-0.00139572  0.16888002]]]
Estimated Priors:
[0.17206659 0.44399981 0.3839336 ]

Converged in 19 iterations
Initial Means:
[[1.6 1.4]
 [1.4 1.6]
...
  [ 0.83672489  0.92191631]]]
Estimated Priors:
[0.35798648 0.64201352]
```

Impact of initial conditions on the results:

Initial condition 1:

The initialization was quite far from the true parameters, leading to estimates that might not align well with the true parameters.

Initial condition 2:

This initialization was somewhat closer, leading to better estimates.

Initial condition 3:

Using fewer components (J=2) than the actual number of underlying distributions (J=3) led to poor approximation, demonstrating the importance of choosing an appropriate number of components.

Impact of the number of components on the results:

When J was set to 2, the model could not capture all the underlying distributions, leading to suboptimal estimates. When J was set to 3, the model performed better in capturing the data distribution.

13.

A) We have two states:

S1: Coin A
S2: Coin B

$$\pi = \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix}, \text{ Transition probabilities} = \begin{bmatrix} 0.4 & 0.6 \\ 0.4 & 0.6 \end{bmatrix}, \text{ Emission probabilities} = \begin{bmatrix} 0.6 & 0.4 \\ 0.4 & 0.6 \end{bmatrix}$$

The columns correspond to probabilities of Heads and Tails.

B)

```python
# 13

import numpy as np
from hmmlearn import hmm

# HMM parameters
states = ["A", "B"]
observations = ["H", "T"]
n_states = len(states)
n_observations = len(observations)

# Initial state probabilities
start_prob = np.array([1.0, 0.0])

# Transition probabilities
transition_matrix = np.array([
    [0.4, 0.6],
    [0.4, 0.6]
])
```

```python
# Emission probabilities
emission_matrix = np.array([
    [0.6, 0.4],  # Coin A: 0.6 heads, 0.4 tails
    [0.4, 0.6]   # Coin B: 0.4 heads, 0.6 tails
])

obs_map = {'H': 0, 'T': 1}
obs_sequence = np.array([obs_map[obs] for obs in ['H', 'H', 'T', 'H', 'T']])


model = hmm.MultinomialHMM(n_components=n_states, n_iter=100)

model.startprob_ = start_prob
model.transmat_ = transition_matrix
model.emissionprob_ = emission_matrix

obs_sequence = obs_sequence.reshape(-1, 1)

model.fit(obs_sequence)

# Log likelihood of the observed sequence
log_likelihood = model.score(obs_sequence)

print(f"Log likelihood of the observed sequence: {log_likelihood}")
```

✓ 0.0s

```
MultinomialHMM has undergone major changes. The previous version was implementing a CategoricalHMM (a special case of Multinom
https://github.com/hmmlearn/hmmlearn/issues/335
https://github.com/hmmlearn/hmmlearn/issues/340
Even though the 'startprob_' attribute is set, it will be overwritten during initialization because 'init_params' contains 's'
Even though the 'transmat_' attribute is set, it will be overwritten during initialization because 'init_params' contains 't'
Log likelihood of the observed sequence: -8.326672684688674e-17
```

0

Log-likelihood of the observed sequence is -8.3266e-17.