# MTH 602 Scientific Machine Learning

Homework 5

11/5/2025

S. M. Mahfuzul Hasan

02181922

UMass Dartmouth

# I. PAPER & PENCIL WORK

**1.** Book problem 5.13:

The likelihood function can be written as,

$$p\left(\{\phi_n, \mathbf{t}_n\} \mid \{\pi_k\}\right) = \prod_{n=1}^{N}\prod_{k=1}^{K}\{p\left(\phi_n|\mathcal{C}_k\right)\pi_k\}^{t_{nk}} \tag{1}$$

Taking `ln` in equation (1),

$$\begin{aligned}\ln\left[p\left(\{\phi_n, \mathbf{t}_n\} \mid \{\pi_k\}\right)\right] &= \ln\left[\prod_{n=1}^{N}\prod_{k=1}^{K}\{p\left(\phi_n|\mathcal{C}_k\right)\pi_k\}^{t_{nk}}\right] \\ &= \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\left[\ln\left(p\left(\phi_n|\mathcal{C}_k\right)\right) + \ln\pi_k\right]\end{aligned} \tag{2}$$

To maximize equation (2) with respect to $\pi_k$, the constraint $\sum\limits_{k=1}^{K}\pi_k = 1$ must be enforced. This can be done introducing a *Lagrange* multiplier. So, we need to maximize

$$\begin{aligned}\ln\left[p\left(\{\phi_n, \mathbf{t}_n\} \mid \{\pi_k\}\right)\right] + \lambda\left(\sum_{k=1}^{K}\pi_k - 1\right) &= \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\left[\ln\left(p\left(\phi_n|\mathcal{C}_k\right)\right) + \ln\pi_k\right] \\ &\quad + \lambda\left(\sum_{k=1}^{K}\pi_k - 1\right)\end{aligned} \tag{3}$$

Now taking derivative of equation (3) and setting it to zero, we get

$$\begin{aligned}&\frac{\partial}{\partial\pi_k}\left(\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\left[\ln\left(p\left(\phi_n|\mathcal{C}_k\right)\right) + \ln\pi_k\right] + \lambda\left(\sum_{k=1}^{K}\pi_k - 1\right)\right) = 0 \\ &\Rightarrow 0 + \sum_{n=1}^{N}\frac{\partial}{\partial\pi_k}\left(t_{nk}\sum_{k=1}^{K}\ln\pi_k\right) + \lambda\frac{\partial}{\partial\pi_k}\left(\sum_{k=1}^{K}\pi_k\right) - 0 = 0 \\ &\Rightarrow \sum_{n=1}^{N}\frac{\partial}{\partial\pi_k}\left(t_{n1}\ln\pi_1 + \ldots.. + t_{nk}\ln\pi_k + \ldots.. + t_{nK}\ln\pi_K\right) + \\ &\quad\lambda\frac{\partial}{\partial\pi_k}\left(\pi_1 + \ldots.. + \pi_k + \ldots.. + \pi_K\right) = 0 \\ &\Rightarrow \frac{\sum\limits_{n=1}^{N} t_{nk}}{\pi_k} + \lambda = 0 \quad \text{[since, } t_{nk} \text{ is a constant]} \\ &\Rightarrow N_k = -\lambda\pi_k \quad \left[\text{since, } \sum_{n=1}^{N} t_{nk} = N_k\right]\end{aligned} \tag{4}$$

Summing equation(4) on both sides with $\sum\limits_{k=1}^{K}$,

$$\begin{aligned}\sum_{k=1}^{K} N_k &= -\lambda \sum_{k=1}^{K} \pi_k \\ \Rightarrow N &= -\lambda \cdot 1 \\ \Rightarrow \lambda &= -N\end{aligned} \tag{5}$$

Plugging $\lambda$ value back in equation (4),

$$\begin{aligned}N_k &= -(-N)\pi_k \\ \Rightarrow \pi_k &= \frac{N_k}{N} \qquad \text{(Showed)}\end{aligned} \tag{6}$$

**2.** Book problem 5.14:

Given,

$$\begin{aligned}p\left(\phi_n | \mathcal{C}_k\right) &= \mathcal{N}\left(\phi_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}\right) \\ &= \frac{1}{2\pi^{D/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\phi_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}\left(\phi_n - \boldsymbol{\mu}_k\right)\right\}\end{aligned} \tag{7}$$

Using equation (7) into equation (2),

$$\begin{aligned}&\ln\left[p\left(\{\phi_n, \mathbf{t}_n\} \mid \{\pi_k\}\right)\right] \\ &= \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\left[\ln\left(\frac{1}{2\pi^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\phi_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}\left(\phi_n - \boldsymbol{\mu}_k\right)\right\}\right) + \ln\pi_k\right] \\ &= \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\left[-\ln(2\pi^{D/2}) - \frac{1}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}(\phi_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}\left(\phi_n - \boldsymbol{\mu}_k\right) + \ln\pi_k\right]\end{aligned} \tag{8}$$

Differentiating equation (8) with respect to $\boldsymbol{\mu}_k$ and setting it to zero, we get

$$\begin{aligned}&\frac{\partial}{\partial \boldsymbol{\mu}_k}\left(\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\left[-\ln(2\pi^{D/2}) - \frac{1}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}(\phi_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}\left(\phi_n - \boldsymbol{\mu}_k\right) + \ln\pi_k\right]\right) = 0 \\ &\Rightarrow -0 - 0 - \sum_{n=1}^{N} t_{nk}\left(\frac{1}{2}\right)(-2)\boldsymbol{\Sigma}^{-1}\left(\phi_n - \boldsymbol{\mu}_k\right) + 0 = 0 \\ &\Rightarrow \boldsymbol{\Sigma}^{-1}\sum_{n=1}^{N} t_{nk}\left(\phi_n - \boldsymbol{\mu}_k\right) = 0 \\ &\Rightarrow \sum_{n=1}^{N} t_{nk}\phi_n = \boldsymbol{\mu}_k \sum_{n=1}^{N} t_{nk} \\ &\Rightarrow \boldsymbol{\mu}_k = \frac{1}{N_k}\sum_{n=1}^{N} t_{nk}\phi_n \qquad \left[\text{since, } \sum_{n=1}^{N} t_{nk} = N_k\right] \\ &\qquad\qquad\qquad \text{(Showed)}\end{aligned} \tag{9}$$

Now, differentiating equation (8) with respect to $\mathbf{\Sigma}^{-1}$ and setting it to zero, we get

$$\frac{\partial}{\partial \mathbf{\Sigma}^{-1}}\left(\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\left[-\ln(2\pi^{D/2}) - \frac{1}{2}\ln|\mathbf{\Sigma}| - \frac{1}{2}(\phi_n - \boldsymbol{\mu}_k)^T\mathbf{\Sigma}^{-1}(\phi_n - \boldsymbol{\mu}_k) + \ln\pi_k\right]\right) = 0$$

$$\Rightarrow -0 - \frac{1}{2}\frac{\partial}{\partial \mathbf{\Sigma}^{-1}}\left(\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\ln|\mathbf{\Sigma}|\right) - \frac{1}{2}\frac{\partial}{\partial \mathbf{\Sigma}^{-1}}\left(\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}(\phi_n - \boldsymbol{\mu}_k)^T\mathbf{\Sigma}^{-1}(\phi_n - \boldsymbol{\mu}_k)\right) + 0 = 0$$

$$\Rightarrow N(-\mathbf{\Sigma}^T) + \frac{\partial}{\partial \mathbf{\Sigma}^{-1}}\left(\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\mathrm{Tr}\left(\mathbf{\Sigma}^{-1}(\phi_n - \boldsymbol{\mu}_k)(\phi_n - \boldsymbol{\mu}_k)^T\right)\right) = 0$$

$$[\text{since, } \mathbf{x}^T A\mathbf{x} = \mathrm{Tr}\left(A\mathbf{x}\mathbf{x}^T\right)]$$

$$\Rightarrow -N\mathbf{\Sigma} + \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\Big((\phi_n - \boldsymbol{\mu}_k)(\phi_n - \boldsymbol{\mu}_k)^T\Big)^T = 0$$

$$[\text{since, } \mathbf{\Sigma}^T = \mathbf{\Sigma}, \text{ and } \frac{\partial}{\partial \mathbf{\Sigma}^{-1}}\left(\mathrm{Tr}\left(\mathbf{\Sigma}^{-1}A\right)\right) = A^T]$$

$$\Rightarrow \mathbf{\Sigma} = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}(\phi_n - \boldsymbol{\mu}_k)(\phi_n - \boldsymbol{\mu}_k)^T$$

$$= \sum_{k=1}^{K}\frac{N_k}{N}\cdot\frac{1}{N_k}\sum_{n=1}^{N} t_{nk}(\phi_n - \boldsymbol{\mu}_k)(\phi_n - \boldsymbol{\mu}_k)^T = \sum_{k=1}^{K}\frac{N_k}{N}\mathbf{S}_k \quad \text{(Showed)}$$

## II. USING GMM AS A GEOMETRIC AND PROBABILISTIC CLASSIFIER

### A. Recap of the previous assignment (context)

Cont'd...

### B. Setup and notation

Cont'd...

### C. Data for this assignment

Ground truth mixture:

$$w = (0.2, 0.8); \quad \mu_1 = (0,0), \ \Sigma_1 = \begin{pmatrix} 1 & 0.8 \\ 0.8 & 1.5 \end{pmatrix}; \quad \mu_2 = (2,2), \ \Sigma_2 = \begin{pmatrix} 1.2 & -0.5 \\ -0.5 & 0.8 \end{pmatrix}$$

For sampling of the data, please refer to listing 1.

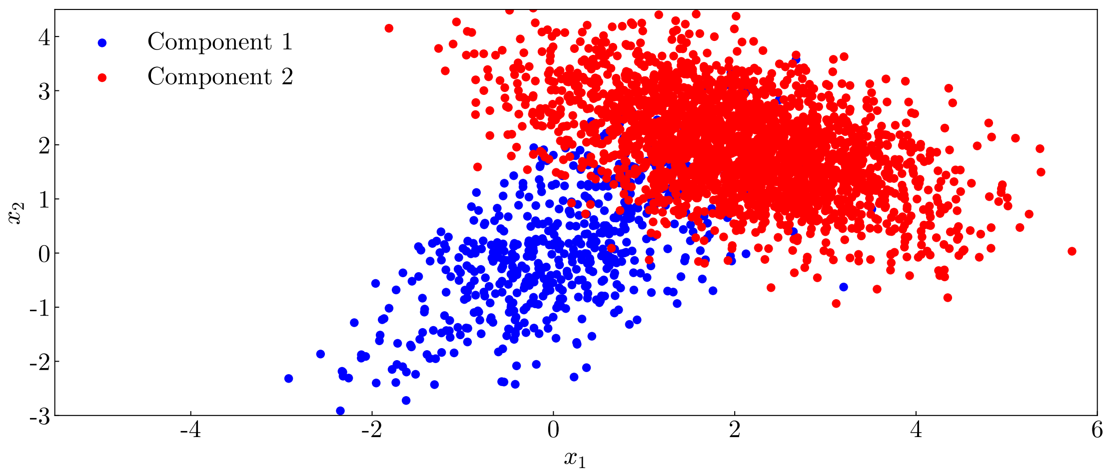


Figure 1: 3000 samples by first sampling the component index, and then sampling $\vec{x} \mid y = k \sim \mathcal{N}(\mu_k, \Sigma_k)$.

### D. Training the Gaussian mixture model

1. Please refer to listing 1 and 2 for the fitted GMM.

   Weights:

   $$\hat{w} = \{0.83249319, 0.16750681\}$$

   Means:

   $$\hat{\mu} = \{(1.99485398, 1.95655549), (-0.07531723, -0.14512162)\}$$

   Covariances:

   $$\hat{\Sigma} = \left\{ \begin{pmatrix} 1.16217774 & -0.46118657 \\ -0.46118657 & 0.78502043 \end{pmatrix}, \begin{pmatrix} 0.86072986 & 0.61000376 \\ 0.61000376 & 1.1706272 \end{pmatrix} \right\}$$

2. Please refer to listing 1 and 2 for mapping.

   Fitted GMM aligned to classes:

$$\hat{w} = (0.16750681, 0.83249319)$$
$$\hat{\mu}_1 = (-0.07531723, -0.14512162), \;\; \hat{\Sigma}_1 = \begin{pmatrix} 0.86072986 & 0.61000376 \\ 0.61000376 & 1.1706272 \end{pmatrix}$$
$$\hat{\mu}_2 = (1.99485398, 1.95655549), \;\; \hat{\Sigma}_2 = \begin{pmatrix} 1.16217774 & -0.46118657 \\ -0.46118657 & 0.78502043 \end{pmatrix}$$

3. Verification of the correctness of the GMM model has been done both quantitatively and qualitatively.

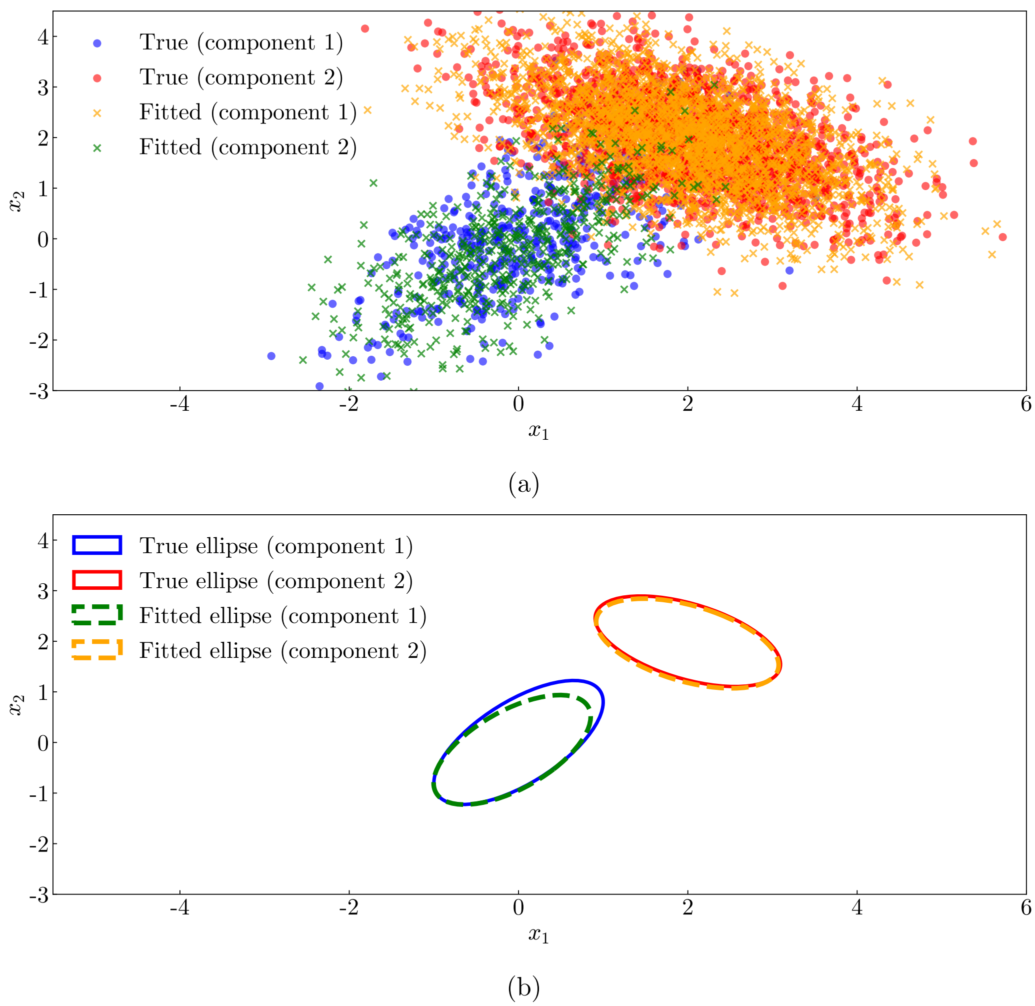


Figure 2: (a) Overlay of fitted samples on the true samples. (b) $1\sigma$ ellipse of $\Sigma$ for fitted and true samples.

Firstly, The $L1$ error norm for the weights and $L2$ error norm for the means and covariances were computed. The reported values are:

$$\|\hat{w}_1 - w_1\| = 0.0325, \quad \|\hat{w}_2 - w_2\| = 0.0325$$
$$\|\hat{\mu}_1 - \mu_1\|_2 = 0.1635, \quad \|\hat{\Sigma}_1 - \Sigma_1\|_2 = 0.4468$$
$$\|\hat{\mu}_2 - \mu_2\|_2 = 0.0437, \quad \|\hat{\Sigma}_2 - \Sigma_2\|_2 = 0.0669$$

The fitted model almost captures the proportions of the mixture, showing 0.0325 absolute error for both the weights. The mean and covariance error for component 1 is higher comparatively than mean and covariance error for component 2. However, that is reasonable since the original weights were 0.2 for component 1, therefore it had very less sampled data points compared to component 2.

Then, the average log-likelihood per sample was calculated both for training and testing data. The values for training and testing data came out to be $-3.0105$ and $-3.0615$, respectively. The higher log-likelihood on the training data indicates that the fitted GMM captures the training distribution well. The slightly lower test value is expected, as unseen data typically yield lower likelihoods. Since, the difference is only 0.05 ($\approx 1.7\%$), this indicates the good generalization behavior of the model, and that it is not prone to overfitting.

The visual check was done as a comparison between the fitted samples and true samples in figure 2. It shows that true and fitted samples span over the same regions. $1\sigma$ ellipses of $\Sigma$ for fitted and true samples are almost exactly the same for component 2, while component 1 has a slight deviation towards right edge between the two ellipses, which can be attributed to the higher $L2$ norm of error $\Sigma_1$. This is, again, due to small sample size of training data for component 1.

Lastly, the prediction accuracy of the model was checked on training and testing data using MAP decision rule. The accuracy shown on training and testing data are 95.12% and 95.33%, respectively which suggest that the model was able to predict seen and unseen data with the same level of accuracy. This is a sign of good generalization of the model. This indicates the capability of the model to separate classes convincingly, barring some miscalssifications due to overlap between the tails of Gaussians, and limited sample size of component 1.

Confusion matrix for training data:

| | Predicted: Class-1 | Predicted: Class-2 |
|---|---|---|
| Actual: Class-1 | 353 | 105 |
| Actual: Class-2 | 12 | 1930 |

Confusion matrix for testing data:

| | Predicted: Class-1 | Predicted: Class-2 |
|---|---|---|
| Actual: Class-1 | 82 | 26 |
| Actual: Class-2 | 2 | 490 |

From the confusion matrices, it is clear that the majority of misclassification happens for component 1.

### E. Geometric classification

1. Decision boundary is reasonable for training data as can be seen from figure 3.

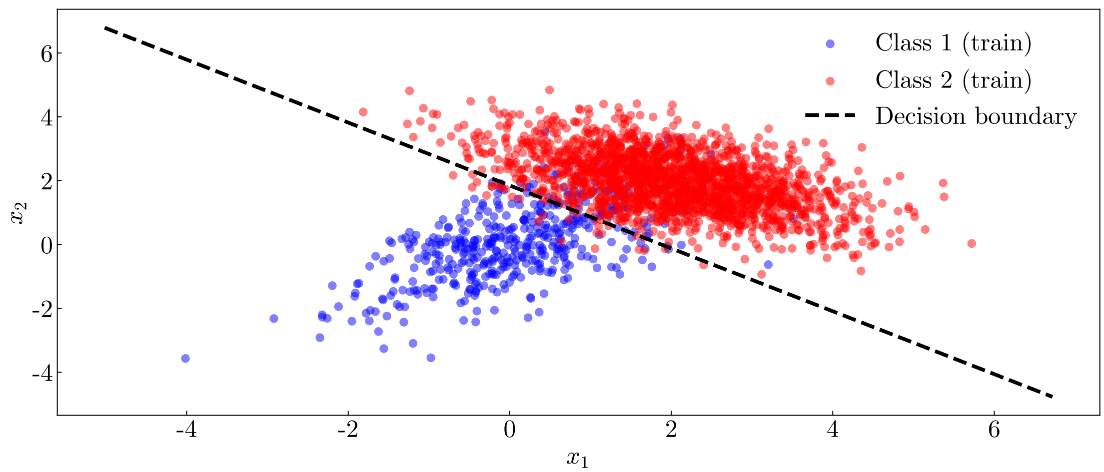


Figure 3: Visual check for training data and boundary.

2. Decision boundary also seems reasonable for testing data as can be seen from figure 4.

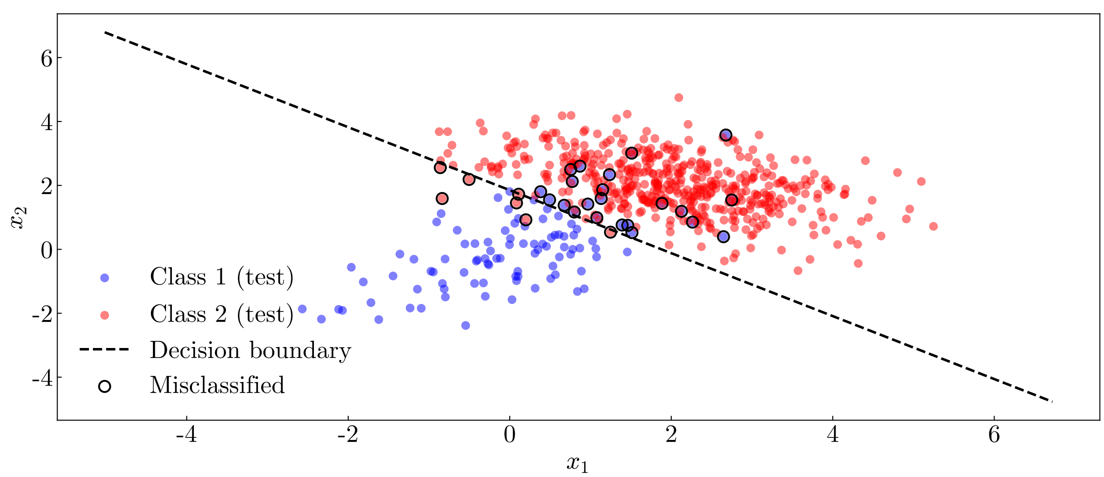


Figure 4: Visual check for testing data, boundary, and misclassifications.

3. Accuracy of the geometric classifier for testing data: 95.17%.

   Confusion matrix:

| | Predicted: Class-1 | Predicted: Class-2 |
|---|---|---|
| Actual: Class-1 | 86 | 22 |
| Actual: Class-2 | 7 | 485 |

### F. Probabilistic GMM classifier with thresholding & ROC

1. Accuracy of the probabilistic classifier for testing data: 95.33%.

   Confusion matrix:

| | Predicted: Class-1 | Predicted: Class-2 |
|---|---|---|
| Actual: Class-1 | 82 | 26 |
| Actual: Class-2 | 2 | 490 |

2. Computed AUC for figure 5: 0.9622

   Both the ROC curve and AUC indicate the very good separability of the classes for the probabilistic classifier. False positive rate increases a bit when True positive rate converges towards 1. However overall, the classifier shows really good capability in identifying the classes.

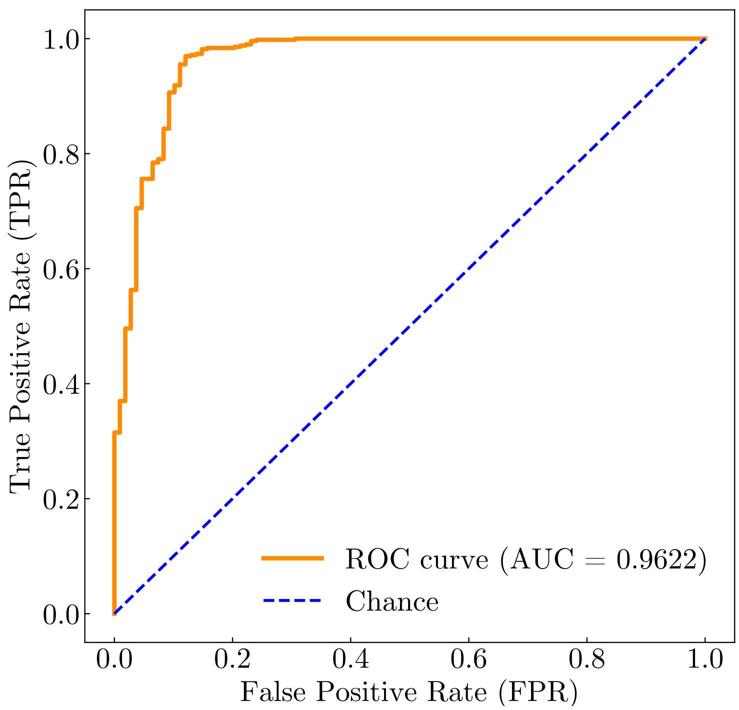


Figure 5: ROC curve.

### G. Comparison: Probabilistic vs geometric classifier

(i) Both classifiers show similar accuracy, but probabilistic classifier shows 0.16% better accuracy compared to geometric classifier. From the confusion matrices of the respective classifiers, and taking the MAP classification from GMM into account, geometric classifier misclassifies class 2 more than probabilistic classifier, where probabilistic classifier is fully in agreement with MAP classification. For a large sample size as class 2, there should be least number of misclassifications for class 2. Geometric classifier shows comparatively bad performance in that case. However, geometric classifier shows better performance for classifying class 1, but since probabilistic classifier relies on posterior probabilities rather than just a fixed perpendicular bisector, it has smoother transition between classes, and higher accuracy overall. Hence, it is slightly better for overlapping regions.

(ii) The dataset constitute a comparatively easy classification task because of highly separable classes due to limited overlap, and thus higher accuracy of the classifiers. High AUC and very good generalization also point to that.

(iii) Class 1 is more difficult to predict for its small sample size due to small weight. This can also be inferred from the confusion matrices, where the misclassifications largely happened for class 1. Class 1 has larger variance, and spreads well into class 2 region, making it difficult for the classifiers to detect them accurately.

```python
from sklearn.mixture import GaussianMixture
from sklearn.metrics import confusion_matrix, accuracy_score, roc_curve, auc
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib.patches import Ellipse

# ===== B. Setup and notation =====
def sample_2d_gaussian(mu, Sigma, n, rng=None):
    if rng is None:
        rng = np.random.default_rng()
    return rng.multivariate_normal(mean=mu, cov=Sigma, size=n)


# ===== C. Data for this assignment =====
# setting parameters
rng = np.random.default_rng(29)
n = 3000

# ground-truth
w_true = np.array([0.2, 0.8])
mu1 = np.array([0.0, 0.0])
Sigma1 = np.array([[1.0, 0.8], [0.8, 1.5]])
mu2 = np.array([2.0, 2.0])
Sigma2 = np.array([[1.2, -0.5], [-0.5, 0.8]])

# initialization
y = rng.choice([1, 2], size=n, p=[0.2, 0.8])
x_ = np.zeros((n, 2))

# sampling
n1 = np.sum(y == 1); n2 = np.sum(y == 2)
x_[y == 1] = sample_2d_gaussian(mu1, Sigma1, n1, rng)
x_[y == 2] = sample_2d_gaussian(mu2, Sigma2, n2, rng)

# parameters for plotting
plt.rcParams["font.family"] = "serif"
plt.rcParams["font.serif"] = ["CMU Serif"]
plt.rcParams["mathtext.fontset"] = "cm"
plt.rcParams["font.size"] = 20
mpl.rcParams["axes.unicode_minus"] = False

# samples plot
```

```python
fig, ax = plt.subplots(figsize=(15, 6))

ax.scatter(x_[y==1, 0], x_[y==1, 1], color="blue", label="Component 1")
ax.scatter(x_[y==2, 0], x_[y==2, 1], color="red", label="Component 2")

plt.xlabel(r'$x_1$')
plt.ylabel(r'$x_2$')
plt.xlim(-5.5,6)
plt.ylim(-3, 4.5)
plt.legend(loc="upper left", frameon=False)
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig(f'x_sample.pdf', dpi=1080)
plt.show()

# 80-20 split
x_train, x_test, y_train, y_test = train_test_split(x_, y, test_size=0.2,
    random_state=29)

# ===== D. Training the Gaussian mixture model =====
# 1. GMM fit
g2 = GaussianMixture(n_components=2, covariance_type='full',
                     reg_covar=1e-6, n_init=10, random_state=29)
g2.fit(x_train)

print("\n1. GMM fit:\n\nFor K=2,")
print("\nWeights:", g2.weights_)
print("\nMeans:", g2.means_)
print("\nCovariances:", g2.covariances_)

# 2. Mapping components to true classes
M = g2.means_
d0 = np.linalg.norm(M[0] - mu1) + np.linalg.norm(M[1] - mu2)
d1 = np.linalg.norm(M[1] - mu1) + np.linalg.norm(M[0] - mu2)
if d0 <= d1:
    class_to_component = {1: 0, 2: 1}
    component_to_class = {0: 1, 1: 2}
else:
    class_to_component = {1: 1, 2: 0}
    component_to_class = {1: 1, 0: 2}

print("\n2. Mapping components to classes:\n\nclass_to_component:",
    class_to_component)
print("component_to_class:", component_to_class)

# reordering components to match classes
w_hat1 = g2.weights_[class_to_component[1]]
w_hat2 = g2.weights_[class_to_component[2]]
mu_hat1 = g2.means_[class_to_component[1]]
mu_hat2 = g2.means_[class_to_component[2]]
Sigma_hat1 = g2.covariances_[class_to_component[1]]
Sigma_hat2 = g2.covariances_[class_to_component[2]]

print("\nTrue and aligned fitted GMM:")
print("\nTrue weights:", w_true)
print("Aligned fitted weights:", [w_hat1, w_hat2])
print("\nTrue means:\n", np.vstack([mu1, mu2]))
print("\nAligned fitted means:\n", np.vstack([mu_hat1, mu_hat2]))
print("\nTrue covariances:")
print(Sigma1, "\n\n", Sigma2)
```

```python
print("\nAligned fitted covariances:")
print(Sigma_hat1, "\n\n", Sigma_hat2)

# 3. Verification of the correctness of the GMM model
# L2 norm of errors
w1_err = abs(w_hat1 - w_true[0])
w2_err = abs(w_hat2 - w_true[1])
mu_err1 = np.linalg.norm(mu_hat1 - mu1, ord=2)
mu_err2 = np.linalg.norm(mu_hat2 - mu2, ord=2)
Sig_err1 = np.linalg.norm(Sigma_hat1 - Sigma1, ord=2)
Sig_err2 = np.linalg.norm(Sigma_hat2 - Sigma2, ord=2)

print(f"\n3. Verification of GMM model:")
print(f"\n||w_1(fit) - w_1(true)|| = {w1_err:.4f}")
print(f"||w_2(fit) - w_2(true)|| = {w2_err:.4f}")
print(f"||mu_1(fit) - mu_1(true)||_2 = {mu_err1:.4f}")
print(f"||mu_2(fit) - mu_2(true)||_2 = {mu_err2:.4f}")
print(f"||Sigma_1(fit) - Sigma_1(true)||_2 = {Sig_err1:.4f}")
print(f"||Sigma_2(fit) - Sigma_2(true)||_2 = {Sig_err2:.4f}")

# average log-likelihood on training and test data
train_logl = g2.score(x_train)
test_logl = g2.score(x_test)

print(f"\nAverage log-likelihood per sample (train): {train_logl:.4f}")
print(f"Average log-likelihood per sample (test):  {test_logl:.4f}")

# Draw samples from the fitted GMM (same number as true data)
x_fit, y_fit = g2.sample(n)

# true and fitted GMMs plot
fig, ax = plt.subplots(figsize=(15, 6))

ax.scatter(x_train[y_train == 1, 0], x_train[y_train == 1, 1],
           color="blue", alpha=0.6, label="True (component 1)")
ax.scatter(x_train[y_train == 2, 0], x_train[y_train == 2, 1],
           color="red", alpha=0.6, label="True (component 2)")

ax.scatter(x_fit[y_fit == 0, 0], x_fit[y_fit == 0, 1], color="orange", alpha=0.7,
           marker='x', label="Fitted (component 1)")
ax.scatter(x_fit[y_fit == 1, 0], x_fit[y_fit == 1, 1], color="green", alpha=0.7,
           marker='x', label="Fitted (component 2)")

plt.xlabel(r"$x_1$")
plt.ylabel(r"$x_2$")
plt.xlim(-5.5,6)
plt.ylim(-3, 4.5)
plt.legend(loc="upper left", frameon=False)
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig(f'x_sample_fit.pdf', dpi=1080)
plt.show()

# 1$\sigma$ ellipse
fig, ax = plt.subplots(figsize=(15, 6))

# true ellipses
for k, (mu_true, Sigma_true, color) in enumerate(
        [(mu1, Sigma1, "blue"), (mu2, Sigma2, "red")], start=1):

```

```python
    # compute eigenvalues & eigenvectors
    eigen_val, eigen_vec = np.linalg.eigh(Sigma_true)

    # sort in descending order
    order = np.argsort(eigen_val)[::-1]
    eigen_val = eigen_val[order]
    eigen_vec = eigen_vec[:, order]

    # rotation angle
    theta = np.degrees(np.arctan2(*eigen_vec[:, 0][::-1])) % 180

    # major and minor axes
    width, height = 2 * np.sqrt(eigen_val)

    # print for reference (optional)
    print(f"\nTrue Component {k}:")
    print(f"  Eigenvalues = {eigen_val}")
    print(f"  Rotation = {theta:.2f} degree")

    ell = Ellipse(xy=mu_true, width=width, height=height, angle=theta, ls="-",
                  edgecolor=color, facecolor='none', lw=3,
                  label=f"True ellipse (component {k})")
    ax.add_patch(ell)

# fitted ellipses
for k, (mu_hat, Sigma_hat, color) in enumerate(
        [(mu_hat1, Sigma_hat1, "green"), (mu_hat2, Sigma_hat2, "orange")], start
            =1):

    # compute eigenvalues & eigenvectors
    eigen_val, eigen_vec = np.linalg.eigh(Sigma_hat)

    # sort in descending order
    order = np.argsort(eigen_val)[::-1]
    eigen_val = eigen_val[order]
    eigen_vec = eigen_vec[:, order]

    # rotation angle
    theta = np.degrees(np.arctan2(*eigen_vec[:, 0][::-1])) % 180

    # major and minor axes
    width, height = 2 * np.sqrt(eigen_val)

    # print for reference (optional)
    print(f"\nFitted Component {k}:")
    print(f"  Eigenvalues = {eigen_val}")
    print(f"  Rotation = {theta:.2f} degree")

    ell = Ellipse(xy=mu_hat, width=width, height=height, angle=theta, ls="--",
                  edgecolor=color, facecolor='none', lw=4,
                  label=f"Fitted ellipse (component {k})")
    ax.add_patch(ell)

plt.xlabel(r"$x_1$")
plt.ylabel(r"$x_2$")
plt.xlim(-5.5, 6)
plt.ylim(-3, 4.5)
plt.legend(loc="upper left", frameon=False)
plt.tick_params(axis="both", which="both", direction="in")
```

```python
plt.savefig(f'ellipse.pdf', dpi=1080)
plt.show()

# training and testing data prediction accuracy
comp_pred_train = g2.predict(x_train)
y_pred_train = np.array([component_to_class[c] for c in comp_pred_train])
train_acc = np.mean(y_pred_train == y_train)

comp_pred_test = g2.predict(x_test)
y_pred_test = np.array([component_to_class[c] for c in comp_pred_test])
test_acc = np.mean(y_pred_test == y_test)

print(f"\nTraining accuracy: {train_acc*100:.2f}%")
print(f"Test accuracy: {test_acc*100:.2f}%")

print("\nConfusion matrix (train):")
print(confusion_matrix(y_train, y_pred_train, labels=[1, 2]))

print("\nConfusion matrix (test):")
print(confusion_matrix(y_test, y_pred_test, labels=[1, 2]))


# ===== E. Geometric classification =====
# 1. Visual check for training data and boundary
# geometric parameters
r_hat = (mu_hat2 - mu_hat1) / np.linalg.norm(mu_hat2 - mu_hat1)
m = 0.5 * (mu_hat1 + mu_hat2)

def classify_geom(x):
    return np.where(np.dot(x - m, r_hat) < 0, 1, 2)

# decision boundary
x_vals = np.linspace(np.min(x_train[:,0]) - 1, np.max(x_train[:,0]) + 1, 200)
y_vals = m[1] - (r_hat[0]/r_hat[1]) * (x_vals - m[0])

# plot training boundary and data
fig, ax = plt.subplots(figsize=(15, 6))
ax.scatter(x_train[y_train==1,0], x_train[y_train==1,1], c='blue',
           alpha=0.5, label='Class 1 (train)')
ax.scatter(x_train[y_train==2,0], x_train[y_train==2,1], c='red',
           alpha=0.5, label='Class 2 (train)')
ax.plot(x_vals, y_vals, 'k--', lw=3, label='Decision boundary')

plt.xlabel(r'$x_1$');
plt.ylabel(r'$x_2$')
plt.legend(loc="upper right", frameon=False)
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig(f'train_bndry.pdf', dpi=1080)
plt.show()

# 2. Visual check for testing data, boundary, and misclassification
# classify testing data
y_geom_test = classify_geom(x_test)

# plot testing decision boundary and data
fig, ax = plt.subplots(figsize=(15, 6))
ax.scatter(x_test[y_test==1,0], x_test[y_test==1,1], c='blue', alpha=0.5,
           label='Class 1 (test)')
ax.scatter(x_test[y_test==2,0], x_test[y_test==2,1], c='red', alpha=0.5,
```

```python
            label='Class 2 (test)')
ax.plot(x_vals, y_vals, 'k--', lw=2, label='Decision boundary')

mis_idx = (y_geom_test != y_test)
ax.scatter(x_test[mis_idx,0], x_test[mis_idx,1], facecolors='none',
            edgecolors='black', lw=1.5, s=80, label='Misclassified')

plt.xlabel(r'$x_1$');
plt.ylabel(r'$x_2$')
plt.legend(loc="lower left", frameon=False)
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig(f'test_bndry.pdf', dpi=1080)
plt.show()

# 3. Confusion matrix (test)
confuse_geom = confusion_matrix(y_test, y_geom_test, labels=[1, 2])
accuracy_geom = accuracy_score(y_test, y_geom_test)
print("\n3. Accuracy and confusion matrix of geometric classifier:\n\nAccuracy (test): {:.2f}%".format(accuracy_geom*100))
print("Confusion matrix (test):")
print(confuse_geom)

# ===== F. Probabilistic classifier with thresholding & ROC =====
# Compute posterior probabilities for each class using the fitted GMM
proba = g2.predict_proba(x_test)

# identify the fitted GMM component corresponding to true class 2
comp_idx_for_class2 = class_to_component[2]

# extract posterior probability
proba_class2 = proba[:, comp_idx_for_class2]

# classification based on threshold
tau = 0.5
y_prob_test = np.where(proba_class2 >= tau, 2, 1)

# 1. Confusion matrix (test)
accuracy_prob = accuracy_score(y_test, y_prob_test)
confuse_prob = confusion_matrix(y_test, y_prob_test, labels=[1, 2])

print(f"\n1. Accuracy and confusion matrix of probabilistic classifier:\n\
nAccuracy (test): {accuracy_prob*100:.2f}%")
print("Confusion matrix (test):")
print(confuse_prob)

# 2. ROC curve and AUC
# convert true labels {1,2} to {0,1}
y_test_b = (y_test == 2).astype(int)

# compute ROC curve points across thresholds
f_p, t_p, thresholds = roc_curve(y_test_b, proba_class2)

# compute AUC
roc_auc = auc(f_p, t_p)

print(f"\n2. ROC curve and AUC:\n\nAUC= {roc_auc:.4f}")

# plot ROC curve
fig, ax = plt.subplots(figsize=(8, 8))
```

```python
ax.plot(f_p, t_p, color='darkorange', lw=3, label=f'ROC curve (AUC = {roc_auc:.4f
    })')
ax.plot([0, 1], [0, 1], color='blue', lw=2, ls='--', label="Chance")

plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(frameon=False)
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig(f'roc.pdf', dpi=1080)
plt.show()
```

Listing 1: *gmm.py*

```
1. GMM fit:

For K=2,

Weights: [0.83249319 0.16750681]

Means: [[ 1.99485398  1.95655549]
 [-0.07531723 -0.14512162]]

Covariances: [[[ 1.16217774 -0.46118657]
  [-0.46118657  0.78502043]]

 [[ 0.86072986  0.61000376]
  [ 0.61000376  1.1706272 ]]]

2. Mapping components to classes:

class_to_component: {1: 1, 2: 0}
component_to_class: {1: 1, 0: 2}

True and aligned fitted GMM:

True weights: [0.2 0.8]
Aligned fitted weights: [0.1675068064372643, 0.8324931935627358]

True means:
 [[0. 0.]
 [2. 2.]]

Aligned fitted means:
 [[-0.07531723 -0.14512162]
 [ 1.99485398  1.95655549]]

True covariances:
[[1.  0.8]
 [0.8 1.5]]

 [[ 1.2 -0.5]
 [-0.5  0.8]]

Aligned fitted covariances:
[[0.86072986 0.61000376]
 [0.61000376 1.1706272 ]]

 [[ 1.16217774 -0.46118657]
 [-0.46118657  0.78502043]]
```

```

3. Verification of GMM model:

||w_1(fit) - w_1(true)|| = 0.0325
||w_2(fit) - w_2(true)|| = 0.0325
||mu_1(fit) - mu_1(true)||_2 = 0.1635
||mu_2(fit) - mu_2(true)||_2 = 0.0437
||Sigma_1(fit) - Sigma_1(true)||_2 = 0.4468
||Sigma_2(fit) - Sigma_2(true)||_2 = 0.0669

Average log-likelihood per sample (train): -3.0105
Average log-likelihood per sample (test):  -3.0615

True Component 1:
  Eigenvalues = [2.08815273 0.41184727]
  Rotation = 53.68 degree

True Component 2:
  Eigenvalues = [1.53851648 0.46148352]
  Rotation = 145.90  degree

Fitted Component 1:
  Eigenvalues = [1.64505416 0.38630291]
  Rotation = 52.13 degree

Fitted Component 2:
  Eigenvalues = [1.47185099 0.47534718]
  Rotation = 146.12 degree

Training accuracy: 95.12%
Test accuracy: 95.33%

Confusion matrix (train):
[[ 353  105]
 [  12 1930]]

Confusion matrix (test):
[[ 82  26]
 [  2 490]]

3. Accuracy and confusion matrix of geometric classifier:

Accuracy (test): 95.17%
Confusion matrix (test):
[[ 86  22]
 [  7 485]]

1. Accuracy and confusion matrix of probabilistic classifier:

Accuracy (test): 95.33%
Confusion matrix (test):
[[ 82  26]
 [  2 490]]

2. ROC curve and AUC:

AUC= 0.9622
```

Listing 2: Output terminal for *gmm.py*

## III. PARTICLE PHYSICS: CLASSIFICATION OF A PARENT PARTICLE FROM TWO-BODY DECAY KINEMATICS

Four physics-informed features were taken into account for the solution of the problem.

Invariant mass:

For $c = 1$,

$$m = \sqrt{(E_1 + E_2)^2 - \left((P_{x1} + P_{x2})^2 + (P_{y1} + P_{y2})^2 + (P_{z1} + P_{z2})^2\right)}$$

Opening angle:

$$cos\theta = \frac{P_{x1} \cdot P_{x2} + P_{y1} \cdot P_{y2} + P_{z1} \cdot P_{z2}}{\sqrt{\left(P_{x1}^2 + P_{y1}^2 + P_{z1}^2\right)\left(P_{x2}^2 + P_{y2}^2 + P_{z2}^2\right)}}$$

Transverse momentum:

$$P_T = \sqrt{(P_{x1} + P_{x2})^2 + (P_{y1} + P_{y2})^2}$$

Energy asymmetry:

$$E_{asym} = \frac{E_1 - E_2}{E_1 + E_2}$$

where, subscripts 1 and 2 refer to daughter 1 and 2, respectively.

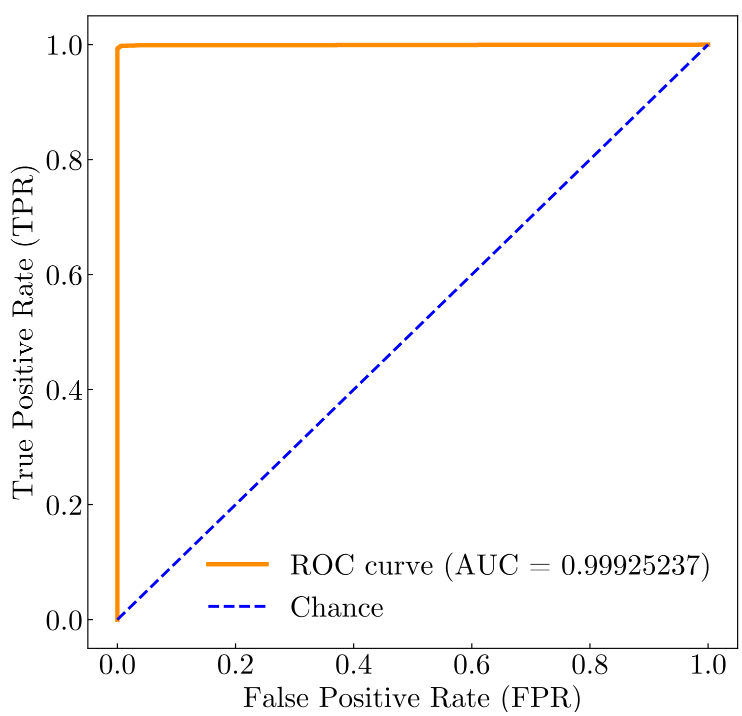


Figure 6: ROC curve for the classifier.

After computing these physical features, the training data were split 80-20 for training and validation. The features were scaled to mean 0 and variance 1. Then, the probabilistic classifier *Logistic*

*Regression* model was used, where

$$p\left(B|\mathbf{x}\right) = \frac{1}{1 + \exp\left(-\phi\left(\mathbf{x}\,;\boldsymbol{\omega}\right)\right)}$$

The accuracy of the model was computed for validation set: 99.60%.

And, the confusion matrix:

| | Predicted: Particle A | Predicted: Particle B |
|---|---|---|
| Actual: Particle A | 1200 | 5 |
| Actual: Particle B | 11 | 2784 |

The prediction performance of the classifier is pretty good. That is further be verified from the ROC curve in figure 6. The AUC is 0.99925237, which is very close to 1.

For the test set, predicted count of parent types:

$$\texttt{N\_A\_pred} = 155 \qquad \texttt{N\_B\_pred} = 345$$

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score,
    roc_curve
import os

# very small constant to avoid division by zero
epsi = 1e-15

# load training data
data = np.genfromtxt("train_hw5.csv", delimiter=",", skip_header=1, dtype=str)

# separate features and labels
x_raw = data[:, :-1].astype(float)      # first 8 columns
labels = data[:, -1]                     # last column

# map A to 0, B to 1
y = np.where(labels == "A", 0, 1)

# unpack columns
E1, px1, py1, pz1, E2, px2, py2, pz2 = [x_raw[:, i] for i in range(8)]

# physics-informed features
E  = E1 + E2 # total energy of the daughters
px = px1 + px2 # total x- momentum
py = py1 + py2 # total y- momentum
pz = pz1 + pz2 # total z- momentum

# total invariant mass
m2 = E**2 - (px**2 + py**2 + pz**2)
m  = np.sqrt(np.clip(m2, 0, None))

```

```python
# opening angle
p1 = np.sqrt(px1**2 + py1**2 + pz1**2)
p2 = np.sqrt(px2**2 + py2**2 + pz2**2)
cos_open = (px1*px2 + py1*py2 + pz1*pz2) / np.clip(p1*p2, epsi, None)

# transverse momentum
p_t = np.sqrt(px**2 + py**2)

# energy asymmetry
E_asym = (E1 - E2) / np.clip(E1 + E2, epsi, None)

# feature matrix
x = np.column_stack([m, p_t, cos_open, E_asym])

# 80-20 split and scaling
x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size=0.2,
                                                      random_state=29, stratify=y)
scaler = StandardScaler()
x_train_s = scaler.fit_transform(x_train)
x_valid_s  = scaler.transform(x_valid)

# logistic regression
clf = LogisticRegression(max_iter=1000, C=2.0)
clf.fit(x_train_s, y_train)

# evaluate validation data
y_pred = clf.predict(x_valid_s)
y_prob = clf.predict_proba(x_valid_s)[:, 1]
acc = accuracy_score(y_valid, y_pred)
cm  = confusion_matrix(y_valid, y_pred)
auc = roc_auc_score(y_valid, y_prob)

print(f"\nAccuracy: {acc*100:.2f}%")
print("\nConfusion matrix:\n", cm)
print(f"\nAUC: {auc:.8f}")

f_p, t_p, _ = roc_curve(y_valid, y_prob)

# ROC plot
fig, ax = plt.subplots(figsize=(8, 8))
ax.plot(f_p, t_p, color='darkorange', lw=3, label=f'ROC curve (AUC = {auc:.8f})')
ax.plot([0, 1], [0, 1], color='blue', lw=2, ls='--', label="Chance")

plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(frameon=False)
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig(f'roc_particle.pdf', dpi=1080)
plt.show()

# predict test data
test_data = np.genfromtxt("test_hw5.csv", delimiter=",", skip_header=1)
E1, px1, py1, pz1, E2, px2, py2, pz2 = [test_data[:, i] for i in range(8)]

E  = E1 + E2; px = px1 + px2; py = py1 + py2; pz = pz1 + pz2
m  = np.sqrt(np.clip(E**2 - (px**2 + py**2 + pz**2), 0, None))
p_t = np.sqrt(px**2 + py**2)
p1 = np.sqrt(px1**2 + py1**2 + pz1**2)
p2 = np.sqrt(px2**2 + py2**2 + pz2**2)
```

```python
cos_open = (px1*px2 + py1*py2 + pz1*pz2) / np.clip(p1*p2, epsi, None)
E_asym = (E1 - E2) / np.clip(E1 + E2, epsi, None)

x_test = np.column_stack([m, p_t, cos_open, E_asym])
x_test_s = scaler.transform(x_test)

y_test_pred = clf.predict(x_test_s)


# probabilistic outputs from the logistic regression
prob = clf.predict_proba(x_test_s)[:, 1]
N_test = len(prob)

# soft counts
N_B_pred_soft = np.sum(prob)
N_A_pred_soft = N_test - N_B_pred_soft

# hard counts
N_B_pred_hard = np.sum(y_test_pred == 1)
N_A_pred_hard = np.sum(y_test_pred == 0)

print("\nPredicted parent counts:")
print(f"\nSoft count:")
print(f"\nN_A_pred = {N_A_pred_soft:.2f}, N_B_pred = {N_B_pred_soft:.2f}")
print(f"\nHard count (threshold 0.5):")
print(f"\nN_A_pred = {N_A_pred_hard}, N_B_pred = {N_B_pred_hard}")
print(f"\nTotal parents in test set: N_test = {N_test}")

y_cd = np.where(y_test_pred == 1, 2, 1).astype(int)

# write labels to text file
if os.path.exists("Mahfuzul_hw5.txt"):
    os.remove("Mahfuzul_hw5.txt")

with open("Mahfuzul_hw5.txt", "w", encoding="utf-8") as f:
    for val in y_cd:
        f.write(f"{val}\n")

print(f"\nSaved Mahfuzul_hw5.txt with {len(y_cd)} predictions.")

# check file
y_check = np.loadtxt("Mahfuzul_hw5.txt", dtype=int)
print(y_check[:10])
```

Listing 3: *particle.py*

```
Accuracy: 99.60%

Confusion matrix:
 [[1200    5]
 [   11 2784]]

AUC: 0.99925237

Predicted parent counts:

Soft count:

N_A_pred = 155.21, N_B_pred = 344.79

Hard count (threshold 0.5):

N_A_pred = 155, N_B_pred = 345

Total parents in test set: N_test = 500

Saved Mahfuzul_hw5.txt with 500 predictions.
[2 2 1 2 2 1 2 2 2 2]
```

Listing 4: Output terminal for *particle.py*