Exercise Sheet 2
Ex 1

1 (a) The decision boundary occurs where $g_A(x) = g_B(x)$.
We will find the locus of such $\underline{x}$.

Note: $\underline{X} \sim \mathcal{N}(\mu, \Sigma) \Rightarrow p(\underline{X} = \underline{x}) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} \exp(-\frac{1}{2}(\underline{x}-\underline{\mu})^T \Sigma^{-1} (\underline{x}-\underline{\mu}))$

$$\Rightarrow \log p(\underline{X}=\underline{x}) = \frac{p}{2}\log 2\pi - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(\underline{x}-\underline{\mu})^T \Sigma^{-1} (\underline{x}-\underline{\mu}).$$

$$\therefore \quad g_A(\underline{x}) = g_B(\underline{x}) \iff \log p(\underline{x} \mid y=A) + \log \pi_A = \log p(\underline{x} \mid y=B) + \log \pi_B$$

$$\iff \log p(\underline{x} \mid y=A) - \log p(\underline{x} \mid y=B) + \log \frac{\pi_A}{\pi_B} = 0$$

$$\iff \frac{1}{2}\log|\Sigma_B| - \frac{1}{2}\log|\Sigma_A| + \frac{1}{2}(\underline{x}-\mu_B)^T \Sigma_B^{-1} (\underline{x}-\mu_B)$$

$$- \frac{1}{2}(\underline{x}-\mu_A)^T \Sigma_A^{-1} (\underline{x}-\mu_A) + \log\frac{\pi_A}{\pi_B} = 0 \quad (*)$$

Simplify one part of $(*)$:

$$(\underline{x}-\mu_B)^T \Sigma_B^{-1} (\underline{x}-\mu_B) - (\underline{x}-\mu_A)^T \Sigma_A^{-1} (\underline{x}-\mu_A)$$

$$= \underline{x}^T(\Sigma_B^{-1} - \Sigma_A^{-1})\underline{x} - \mu_B^T \Sigma_B^{-1} \underline{x} + \mu_B^T \Sigma_B^{-1} \mu_B - \underline{x}^T \Sigma_B^{-1} \mu_B$$

$$+ \mu_A^T \Sigma_A^{-1} \underline{x} - \mu_A^T \Sigma_A^{-1} \mu_A + \underline{x}^T \Sigma_A^{-1} \mu_A \quad (**)$$

Note that $\underline{x}^T \Sigma_A^{-1} \mu_A$ is a scalar, and Σ_A is symmetric $\Rightarrow \Sigma_A^{-1}$ is symmetric $[\Sigma_A^{-1} = \Sigma_A^{-1\,T}]$ and $c^T = c$ for any scalar c. Thus

$$\underline{x}^T \Sigma_A^{-1} \mu_A = (\underline{x}^T \Sigma_A^{-1} \mu_A)^T = \mu_A^T \Sigma_A^{-1} \underline{x}.$$

And equiv. for B

$\therefore$ $(**)$ = $\underline{x}^T(\Sigma_B^{-1} - \Sigma_A^{-1})\underline{x} - 2\mu_B^T\Sigma_B^{-1}\underline{x}$

$+ 2\mu_A^T\Sigma_A^{-1}\underline{x} + \mu_B^T\Sigma_B^{-1}\mu_B - \mu_A^T\Sigma_A^{-1}\mu_A$

Substituting into $(*)$:

$\frac{1}{2}\underline{x}^T(\Sigma_B^{-1} - \Sigma_A^{-1})\underline{x}$

$+ (\mu_A^T\Sigma_A^{-1} - \mu_B^T\Sigma_B^{-1})\underline{x}$

$+\frac{1}{2}(\mu_B^T\Sigma_B^{-1}\mu_B) - \frac{1}{2}\mu_A^T\Sigma_A^{-1}\mu_A + \frac{1}{2}\log|\Sigma_B| - \frac{1}{2}\log|\Sigma_A|$

$+ \log\frac{\pi_A}{\pi_B} = 0$

So $\boxed{\Lambda = \frac{1}{2}(\Sigma_B^{-1} - \Sigma_A^{-1})}$

$\underline{w}^T = \mu_A^T\Sigma_A^{-1} - \mu_B^T\Sigma_B^{-1} \Rightarrow \boxed{\underline{w} = \Sigma_A^{-1}\mu_A - \Sigma_B^{-1}\mu_B}$

$\boxed{b = \frac{1}{2}[\mu_B^T\Sigma_B^{-1}\mu_B - \mu_A^T\Sigma_A^{-1}\mu_A + \log|\Sigma_B| - \log|\Sigma_A|] + \log\frac{\pi_A}{\pi_B}}$

Brief explanation:

Here, linear is used to mean affine, ie linear up to a translating constant (in this case b). A linear function f satisfies $f(\lambda\underline{x}) = \lambda f(\underline{x})$. If $f(\underline{x}) = \underline{w}^T\underline{x}$, then $f(\lambda\underline{x}) = \underline{w}^T(\lambda\underline{x}) = \lambda\underline{w}^T\underline{x} = \lambda f(\underline{x})$, ie linearity fulfilled. However for a component $c(\underline{x}) = \underline{x}^T\Lambda\underline{x}$ (ie a quadratic term)
$c(\lambda\underline{x}) = (\lambda\underline{x})^T\Lambda(\lambda\underline{x}) = \lambda^2\underline{x}^T\Lambda\underline{x} = \lambda^2 c(\underline{x})$, ie quadratic scaling, __not__ linear.

b) In this case, we have

$$\Lambda = \tfrac{1}{2}(\Sigma^{-1} - \Sigma^{-1}) = 0$$

$$\underline{w} = \Sigma^{-1}\mu_A - \Sigma^{-1}\mu_B = \Sigma^{-1}(\mu_A - \mu_B)$$

$$b = \tfrac{1}{2}\left[\mu_B^T \Sigma^{-1}\mu_B - \mu_A^T \Sigma^{-1}\mu_A + 2\log|\Sigma| + \log\frac{\pi_A}{\pi_B}\right].$$

In particular, $\Lambda = 0$, so we recover

$$\underline{w}^T \underline{x} + b = 0,$$

as desired.

2. ~~a)~~ See notebook

Machine Learning Essentials SS25 - Exercise Sheet 2

Instructions

- `TODO` 's indicate where you need to complete the implementations.
- You may use external resources, but **write your own solutions**.
- Provide concise, but comprehensible comments to explain what your code does.
- Code that's unnecessarily extensive and/or not well commented will not be scored.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from scipy.stats import multivariate_normal
np.random.seed(42)
```

Exercise 1 - QDA

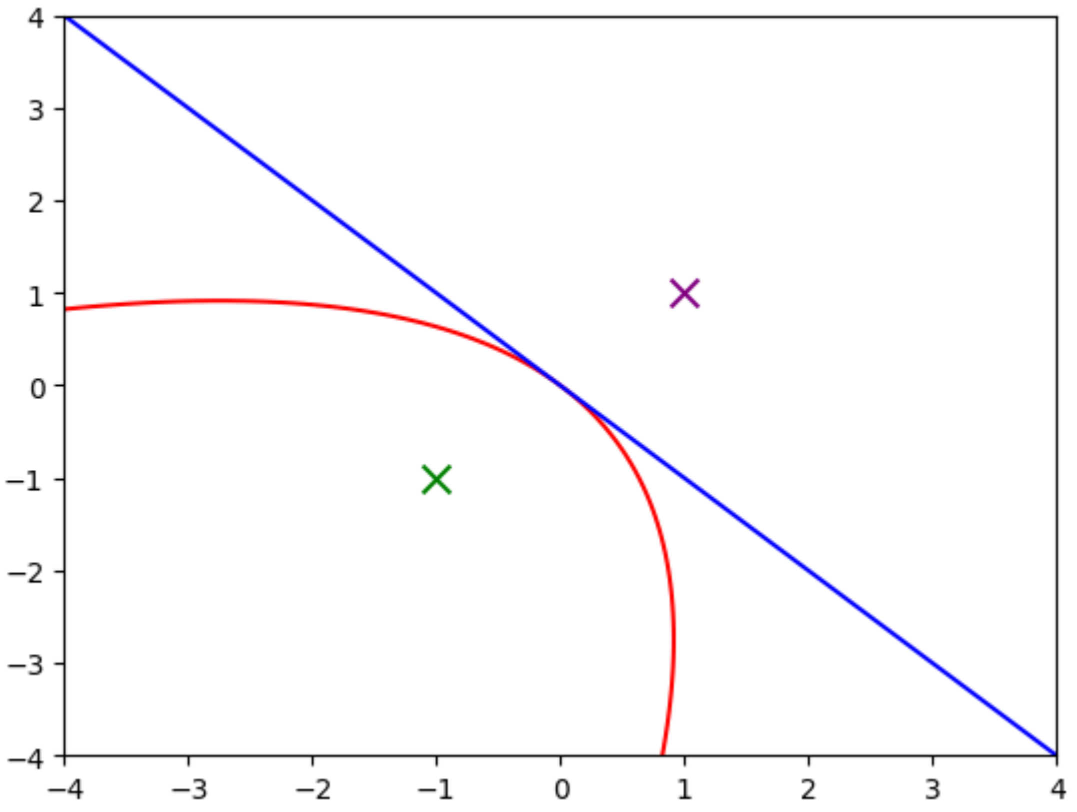
Task 2

In [2]:
```
mu_a = np.array([-1, -1])
mu_b = np.array([1, 1])

sigma_a = np.array([[1, 0.3], [0.3, 1]])
sigma_b = np.array([[1.5, -0.2], [-0.2, 1.5]])

pooled_sigma = 1/2 * (sigma_a + sigma_b)
```

In [3]:
```
# compute the qda boundary
def qda_boundary(x, mu_a, mu_b, sigma_a, sigma_b):
    inv_sigma_a = np.linalg.inv(sigma_a)
    inv_sigma_b = np.linalg.inv(sigma_b)

    term1 = 0.5 * x.T @ (inv_sigma_b - inv_sigma_a) @ x
    w = (inv_sigma_a @ mu_a - inv_sigma_b @ mu_b).T @ x
    b = 0.5 * (mu_b.T @ inv_sigma_b @ mu_b - mu_a.T @ inv_sigma_a @ mu_a)
    return term1 + w + b

# compute the lda boundary
def lda_boundary(x, mu_a, mu_b, sigma):
    inv_sigma = np.linalg.inv(sigma)
    w = (inv_sigma @ (mu_a - mu_b)).T @ x
    b = 0.5 * (mu_b.T @ inv_sigma @ mu_b - mu_a.T @ inv_sigma @ mu_a)

    return w + b
```

In [4]:
```
# Plotting
x = np.linspace(-4, 4, 100)
```

```
y = np.linspace(-4, 4, 100)
X, Y = np.meshgrid(x, y)
grid = np.c_[X.ravel(), Y.ravel()]

Z_qda = np.array([qda_boundary(p, mu_a, mu_b, sigma_a, sigma_b) for p in
Z_lda = np.array([lda_boundary(p, mu_a, mu_b, pooled_sigma) for p in grid

plt.contour(X, Y, Z_qda, levels=[0], colors='r')
plt.contour(X, Y, Z_lda, levels=[0], colors='b')
plt.scatter(mu_a[0], mu_a[1], c='green', marker='x', s=100)
plt.scatter(mu_b[0], mu_b[1], c='purple', marker='x', s=100)
plt.show()
```

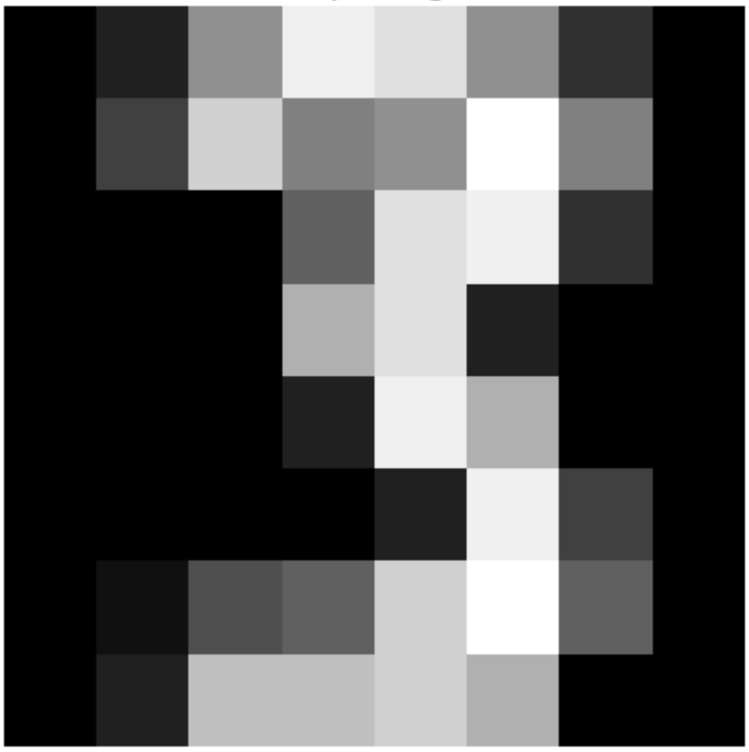


Exercise 2 - Implementing LDA

Task 1

In [21]:
```
# TODO: Load digits dataset, visualize one example image of digit 3
digits = load_digits()

index = np.where(digits.target == 3)[0][1]
example_digit = digits.images[index]


plt.imshow(example_digit, cmap='gray', interpolation='none')
# Interpolation parameter doesn't make pixel values directly visible; we
plt.title(f"Example Digit: {digits.target[0]}")
plt.axis('off')
plt.show()
```

Example Digit: 0



Task 2

In [22]:
```
# TODO: Filter the dataset to keep only digits 3 and 9, split into traini
mask_3 = digits.target == 3
mask_9 = digits.target == 9

filtered_data = digits.data[mask_3 | mask_9]
filtered_target = digits.target[mask_3 | mask_9]

X_train, X_test, y_train, y_test = train_test_split(filtered_data, filter
```

Task 3

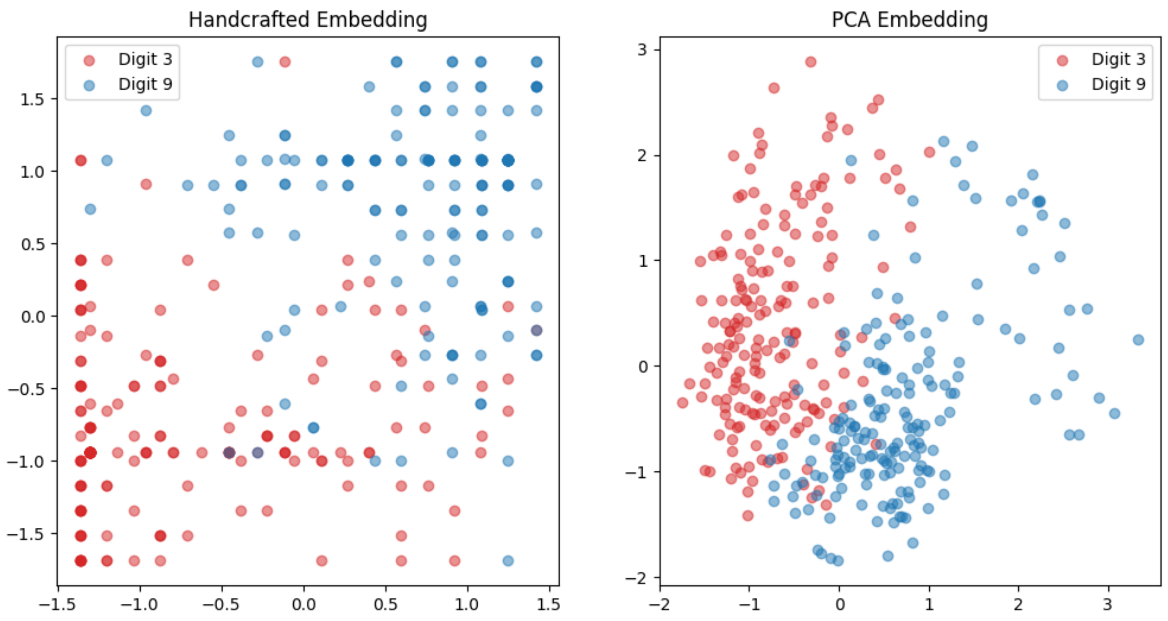
In [29]:
```
def features_2d(data, labels):
    """
    This function takes the 64x1 feature vectors and returns a 2D represe
    """
    digit_3 = data[labels == 3]
    digit_9 = data[labels == 9]

    avg_3 = np.mean(digit_3, axis=0)
    avg_9 = np.mean(digit_9, axis=0)

    difference = avg_3 - avg_9

    # take the two largest values of the difference
    # and use these pixels to create a 2D representation
    largest_indices = np.argsort(np.abs(difference))[-2:]

    features_2d = (data[:, largest_indices], labels)

    return features_2d
```

```
# TODO: Create an embedded dataset, provide a brief justification for you
data_2d, targets_2d = features_2d(X_train, y_train)
test_2d, test_targets_2d = features_2d(X_test, y_test)

# Standardise
data_2d = (data_2d - np.mean(data_2d, axis=0)) / np.std(data_2d, axis=0)
test_2d = (test_2d - np.mean(test_2d, axis=0)) / np.std(test_2d, axis=0)
```

Justification: We require a two dimensional embedding, so we take two of the 64 features: specifically the two in which the class averages differ most. Assuming low variance in these features relative to the differences in averages, then these features should separate the classes well. A more advanced method would be to take the two features which differ most relative to their variance, but we thought this was out of scope.

Task 4

In [43]:
```
def pca_rep(x):
    """
    This function takes the 64x1 feature vectors and returns a 2D represe
    """
    # Standardize the data
    pca = PCA(n_components=2)
    return pca.fit_transform(x)

# TODO: Create a PCA-embedded dataset. Visualize & compare the embeddings
pca_data = pca_rep(X_train)
pca_test = pca_rep(X_test)

# Standardise
pca_data = (pca_data - np.mean(pca_data, axis=0)) / np.std(pca_data, axis
pca_test = (pca_test - np.mean(pca_test, axis=0)) / np.std(pca_test, axis

# Plot
data_all = np.vstack([data_2d, test_2d])
targets_all = np.hstack([targets_2d, test_targets_2d])
pca_all = np.vstack([pca_data, pca_test])

# …existing code above…
plt.figure(figsize=(12, 6))

# Hand-crafted embedding
plt.subplot(1, 2, 1)
mask3 = targets_all == 3
mask9 = targets_all == 9
plt.scatter(data_all[mask3,0], data_all[mask3,1], c='tab:red',   label='D
plt.scatter(data_all[mask9,0], data_all[mask9,1], c='tab:blue',  label='D
plt.title('Handcrafted Embedding')
plt.legend()

# PCA embedding
plt.subplot(1, 2, 2)
plt.scatter(pca_all[mask3,0], pca_all[mask3,1], c='tab:red',  label='Digi
plt.scatter(pca_all[mask9,0], pca_all[mask9,1], c='tab:blue', label='Digi
plt.title('PCA Embedding')
plt.legend()
plt.show()
```



Both embeddings show a clear separation of the two classes. In fact, the PCA separation doesn't seem clearly much better. This could make sense if much of the variance is explained by those two features.

Task 5

In [9]:
```
def fit_lda(training_features, training_labels):
    """
    Compute LDA parameters.
    """
    # filter features with low variance
    #mask = np.var(training_features, axis=0) > 0.001
    #training_features = training_features[:, mask]

    mu = []
    covmat = []
    p = []
    for i in np.unique(training_labels):
        features_pc = training_features[training_labels==i]
        # mu calculation
        mu_pc = np.mean(features_pc, axis=0)
        # cov calculation
        features_pcc = (features_pc - mu_pc)
        covmat_prod = features_pcc.T @ features_pcc
        mu.append(mu_pc)
        covmat.append(covmat_prod)

        # prior calculation
        p.append(features_pc.shape[0] / training_features.shape[0])

    mu = np.array(mu)

    covmat = np.sum(covmat, axis=0) / training_features.shape[0]
    p = np.array(p)
    return mu, covmat, p
```

```
# TODO: Fit seperate LDA models using your hand-crafted embedding, the PC
lda_2d = fit_lda(data_2d, targets_2d)

lda_pca = fit_lda(pca_data, y_train)

# filter data dimensions with less than 0.001 variance
X_train = X_train[:, np.var(X_train, axis=0) > 0.001]
X_test = X_test[:, np.var(X_test, axis=0) > 0.001]
lda_org = fit_lda(X_train, y_train)
```

Task 6

In [10]:
```
lda_2d_mu, lda_2d_covmat, lda_2d_p = lda_2d
lda_pca_mu, lda_pca_covmat, lda_pca_p = lda_pca
lda_org_mu, lda_org_covmat, lda_org_p = lda_org
```

In [11]:
```
print(lda_2d_mu.shape)
```

```
(2, 2)
```

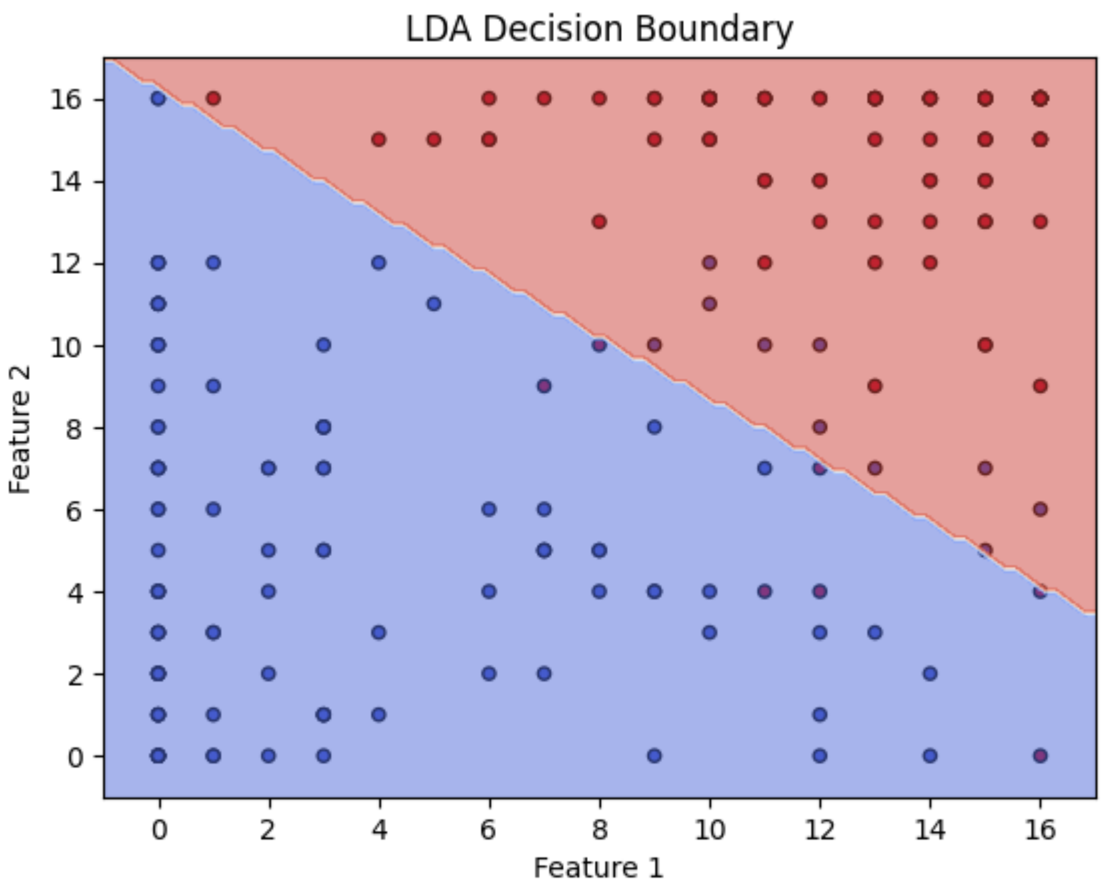
In [12]:
```
def predict_lda(mu, covmat, p, test_features):
    """
    Predict labels using the LDA decision rule.
    """
    # TODO: Implement the LDA decision rule
    covmat_inv = np.linalg.inv(covmat)
    w = covmat_inv @ (mu[1, :] - mu[0, :])
    b = -.5 * (mu[1, :] @ covmat_inv @ mu[1, :] - mu[0, :] @ covmat_inv @
    predicted_labels = np.zeros(test_features.shape[0])
    for i in range(test_features.shape[0]):
        predicted_labels[i] = 1 if (w.T @ test_features[i, :] + b) > 0 el

    return predicted_labels

# TODO: Perform LDA on the filtered train sets of all 3 embeddings, evalu
lda_2d_pred = predict_lda(lda_2d_mu, lda_2d_covmat, lda_2d_p, data_2d)
lda_pca_pred = predict_lda(lda_pca_mu, lda_pca_covmat, lda_pca_p, pca_dat
lda_org_pred = predict_lda(lda_org_mu, lda_org_covmat, lda_org_p, X_train


# report the error rates
def error_rate(predicted_labels, true_labels):
    # replace the values of 0 with 3 and 1 with 9
    predicted_labels[predicted_labels == 0] = 3
    predicted_labels[predicted_labels == 1] = 9
    return np.mean(predicted_labels != true_labels)

error_2d = error_rate(lda_2d_pred, targets_2d)
error_pca = error_rate(lda_pca_pred, y_train)
error_org = error_rate(lda_org_pred, y_train)
print(f"Error rate for original data: {error_org:.2f}")
print(f"Error rate for PCA embedding: {error_pca:.2f}")
print(f"Error rate for 2D embedding: {error_2d:.2f}")

lda_2d_test = predict_lda(lda_2d_mu, lda_2d_covmat, lda_2d_p, test_2d)
lda_pca_test = predict_lda(lda_pca_mu, lda_pca_covmat, lda_pca_p, pca_tes
lda_org_test = predict_lda(lda_org_mu, lda_org_covmat, lda_org_p, X_test)

error_2d_test = error_rate(lda_2d_test, test_targets_2d)
```

```
error_pca_test = error_rate(lda_pca_test, test_targets_2d)
error_org_test = error_rate(lda_org_test, test_targets_2d)
print(f"Test Error rate for original data: {error_org_test:.2f}")
print(f"Test Error rate for PCA embedding: {error_pca_test:.2f}")
print(f"Test Error rate for 2D embedding: {error_2d_test:.2f}")
```

```
Error rate for original data: 0.00
Error rate for PCA embedding: 0.04
Error rate for 2D embedding: 0.08
Test Error rate for original data: 0.40
Test Error rate for PCA embedding: 0.17
Test Error rate for 2D embedding: 0.19
```

Task 7

In [13]:
```
# TODO: For your hand-crafted embedding, visualize the decision boundary

def plot_decision_boundary(mu, covmat, p, data, labels):

    # scatter plot
    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='coolwarm', edgeco

    # min and max vals for the grid
    x_min, x_max = data[:, 0].min() - 1, data[:, 0].max() + 1
    y_min, y_max = data[:, 1].min() - 1, data[:, 1].max() + 1

    # grid for the predictions and boundary
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_mi
    grid_points = np.c_[xx.ravel(), yy.ravel()]
    # throw grid into the LDA function to get predictions
    grid_predictions = predict_lda(mu, covmat, p, grid_points)
    grid_predictions = grid_predictions.reshape(xx.shape)

    # plot the decision boundary
    plt.contourf(xx, yy, grid_predictions, alpha=0.5, cmap='coolwarm')

    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.title('LDA Decision Boundary')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()

plot_decision_boundary(lda_2d_mu, lda_2d_covmat, lda_2d_p, data_2d, targe
```



Task 8

In [44]:
```
def cross_val_lda(X, y, n_splits):
    """
    Perform n-fold cross-validation for LDA using the earlier defined fun
    """
    # TODO: Implement cross-validation for LDA.
    fold_size = len(X) // n_splits
    indices = np.arange(len(X))
    np.random.shuffle(indices)
    folds = np.array_split(indices, n_splits)
    avg_error = []
    for i in range(n_splits):
        test_indices = folds[i]
        train_indices = np.concatenate([folds[j] for j in range(n_splits)

        # filter data dimensions with less than 0.001 variance
        X_train, X_test = X[train_indices, :], X[test_indices, :]
        y_train, y_test = y[train_indices], y[test_indices]

        # filter data dimensions with less than 0.001 variance
        var_indices = np.var(X_train, axis=0) > 0.001
        X_train = X_train[:, var_indices]
        X_test = X_test[:, var_indices]

        mu, covmat, p = fit_lda(X_train, y_train)
        predictions = predict_lda(mu, covmat, p, X_test)

        error = error_rate(predictions, y_test)
        avg_error.append(error)
    avg = np.mean(avg_error)
    return avg, np.std(avg_error)/np.sqrt(n_splits)

# TODO: Perform 10-fold CV on the original data. Report average test erro

n_splits = 10
average, standard_error = cross_val_lda(filtered_data, filtered_target, n

print(f"Average error: {average:.2f}")
print(f"Standard error of error estimates: {standard_error:.2f}")
```

```
Average error: 0.06
Standard error of error estimates: 0.05
```

Compared to the error from task 2.6 trained on the entire data set the average test error rate is much lower with 10 fold cross validation.

Exercise 3 - Statistical Darts

Task 1

In [15]:
```
def simulate_data(mu_true, Sigma_true, n_samples):
    # TODO: Simulate data from a bivariate Gaussian distribution given th

    # Sample
    data = np.random.multivariate_normal(mu_true, Sigma_true, n_samples)

    return data
```

Task 2

In [16]:
```
def compute_mle(data):
    # TODO: Compute the MLE for the mean of a Gaussian distribution.
    mu_mle = np.mean(data, axis=0)
    return mu_mle
```

Task 3

In [17]:
```
def compute_posterior(data, prior, Sigma_true):
    # TODO: Compute the parameters of the posterior distribution for the
    sigma_inv = np.linalg.inv(Sigma_true)

    sigma_prior_inv = np.linalg.inv(prior['Sigma0'])
    mu_prior = prior['mu0']

    mu_mle = compute_mle(data)

    n = data.shape[0]

    Sigma_post = np.linalg.inv(sigma_prior_inv + n * sigma_inv)
    mu_post = Sigma_post @ (sigma_prior_inv @ mu_prior + n * sigma_inv @

    return mu_post, Sigma_post
```

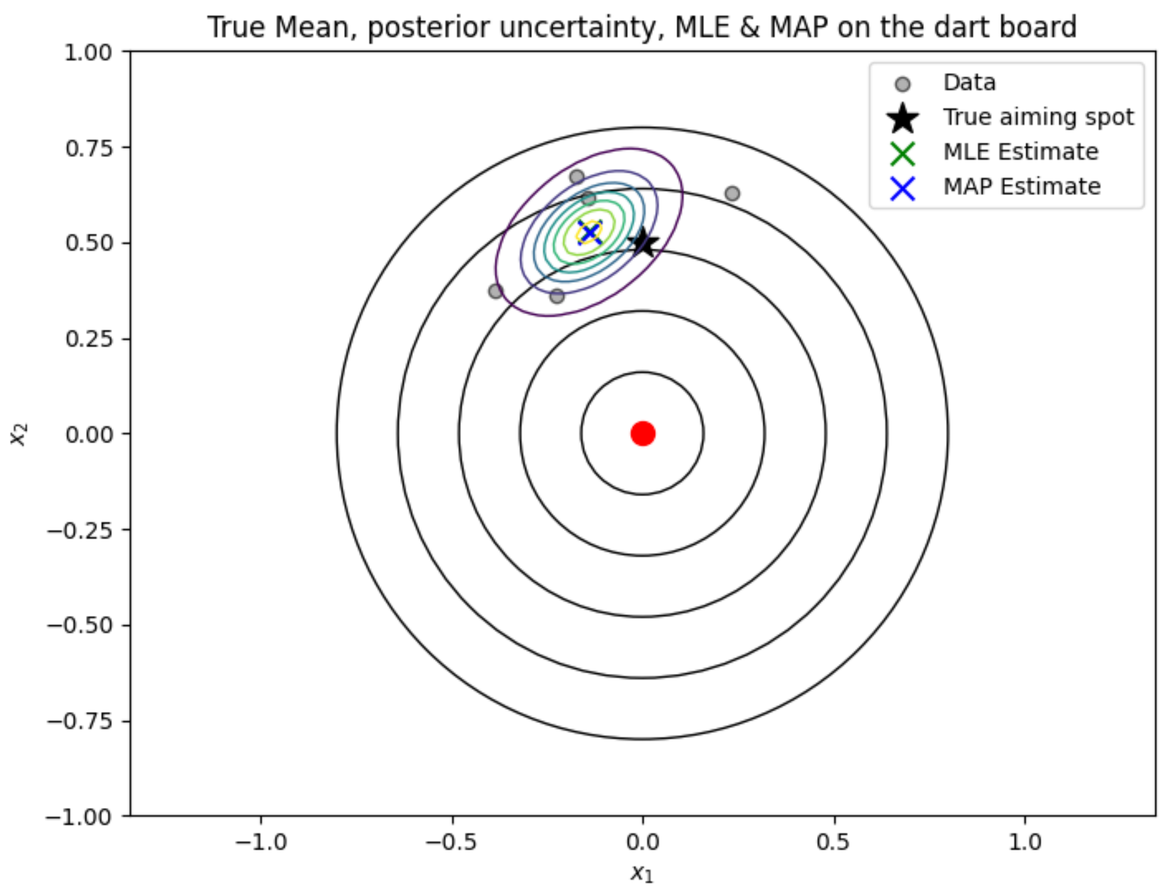
```
def compute_map(data, prior, Sigma_true):
    # TODO: Assign mean of the posterior to mu_map.
    mu_map, _ = compute_posterior(data, prior, Sigma_true)
    return mu_map
```

Task 4

In [45]:
```
def visualize_inference(mu_true, mu_mle, mu_map, mu_post, Sigma_post, dat
                        grid_limits=[-1, 1, -1, 1], n_points=100):
    """
    Visualizes the full posterior distribution as Gaussian isocontours ov
    alongside the true mean, MLE estimate, MAP estimate and the simulated

    Additional parameters:
        grid_limits: [xmin, xmax, ymin, ymax] limits for the 2D grid.
        n_points: Number of grid points per axis.
    """

    # Define the grid
    xmin, xmax, ymin, ymax = grid_limits
    x = np.linspace(xmin, xmax, n_points)
    y = np.linspace(ymin, ymax, n_points)
    X, Y = np.meshgrid(x, y)
    pos = np.dstack((X, Y))

    # Get the posterior distribution
    rv = multivariate_normal(mu_post, Sigma_post)
    # Evaluate the pdf of the posterior @ the grid points
    Z = rv.pdf(pos)

    # Compute some contour levels
    levels = np.linspace(Z.max()*0.05, Z.max()*0.95, 7)

    plt.figure(figsize=(8, 6), facecolor='white')

    # Plot a dartboard-like background (concentric circles)
    center = [0,0]
    radius = 0.8
    for r in [radius, radius*0.8, radius*0.6, radius*0.4, radius*0.2]:
        circle = plt.Circle(center, r, fill=False, color='black')
        plt.gca().add_artist(circle)
    plt.axis('equal')

    # Add bullseye
    plt.plot(center[0], center[1], 'o', markersize=10, c='red')

    # Plot isocontours of  posterior
    contour = plt.contour(X, Y, Z, levels=levels, cmap='viridis',linewidt

    # Add labels to the isocontours (off by default for visibility)
    # plt.clabel(contour, inline=True, fontsize=8, fmt="%.1f")

    # Plot observed data points
    plt.scatter(data[:, 0], data[:, 1], c='gray', edgecolor='k', alpha=0.

    # Plot true mean (ground truth)
    plt.scatter(mu_true[0], mu_true[1], c='black', marker='*', s=200, lab

    # Plot MLE estimate
    plt.scatter(mu_mle[0], mu_mle[1], c='green', marker='x', s=100, label

    # Plot MAP estimate
    plt.scatter(mu_map[0], mu_map[1], c='blue', marker='x', s=100, label=

    plt.title("True Mean, posterior uncertainty, MLE & MAP on the dart bo
    plt.xlabel("$x_1$")
    plt.ylabel("$x_2$")
    plt.legend()
    plt.grid(False)

    plt.show()
```

In [46]:

```
# Ground truth parameters for the dart throws:
mu_true = np.array([0, 0.50])
Sigma_true = np.array([[0.05, 0.02],
                       [0.02, 0.04]])

# Prior for mu - standard normal around the bullseye
prior = {
    "mu0": np.array([0, 0]),
    "Sigma0": np.eye(2)
}

# TODO: Simulate data, compute MLE, MAP and posterior
n_samples = 5
data = simulate_data(mu_true, Sigma_true, n_samples=n_samples)
mu_mle = compute_mle(data)

mu_map = compute_map(data, prior, Sigma_true)
mu_post, Sigma_post = compute_posterior(data, prior, Sigma_true)



# Visualize the inference
visualize_inference(mu_true, mu_mle, mu_map, mu_post, Sigma_post, data)
print(f"MLE estimate for N={n_samples}:", mu_mle)
print(f"MAP estimate for N={n_samples}:", mu_map)
print(f"Posterior covariance for N={n_samples}:\n", Sigma_post)

# TODO: Assess results (see exercise sheet)
```



```
MLE estimate for N=5: [-0.13875311  0.52952618]
MAP estimate for N=5: [-0.139462    0.52587701]
Posterior covariance for N=5:
 [[0.00988543 0.00392903]
 [0.00392903 0.00792092]]
```

Task 5

The posterior reflects a weighted combination of the prior and the likelihood. The posterior variance is quite easy to interpret in 1D, in particular:

$$\sigma_{post}^2 = \frac{1}{\frac{1}{\sigma_0^2} + \frac{N}{\sigma_{true}^2}}$$

which is a weighted harmonic mean, weighted by the number of samples. The harmonic mean works with inverse ($\frac{1}{\sigma_0^2}$) Thus with a more informative prior (smaller σ_0^2 -> larger $1/\sigma_0^2$) the influence on the posterior increases, while for less informative prior (larger σ^2 -> smaller $1/\sigma_0^2$) the influence is smaller.

On the other hand, we can interpret μ_{post} by plugging in σ_{post}^2:

$$\mu_{post} = \frac{P_0}{P_0 + NP_{true}}\mu_0 + \frac{NP_{true}}{P_0 + NP_{true}}\mu_{MLE}$$

which is a weighted average (the usual average) depending on the prior precision P_0 and the posterior precision P_{true}, with precisions being inverse variances.

Task 6

Two conditions under which the MLE and the MAP estimates match:

1. Prior with infinite variance: then the prior is 'uninformative' and thus has no influence on the posterior estimate, and the estimate is generated entirely from the data term. Then $P_0 = 0$ and plugging into

$$\mu_{post} = \frac{P_0}{P_0 + NP_{true}}\mu_0 + \frac{NP_{true}}{P_0 + NP_{true}}\mu_{MLE}$$

one obtains

$$\mu_{post} = \frac{NP_{true}}{NP_{true}}\mu_{MLE} = \mu_{MLE}$$

w. The prior mean matches the MLE estimate (which could not be organised prior to an experiment). Then the same expression simplifies to

$$\mu_{post} = \frac{P_0 + NP_{true}}{P_0 + NP_{true}}\mu_{MLE} = \mu_{MLE}$$

Task 7

1. Suppose you want to estimate the efficacy of a medical treatment (eg in change in probability of survival). Then a positive estimate would sound good, but if there's a high variance, it would indicate that the treatment is ineffective, or even harmful for some patients. This would be very important to know when communicating to patients to allow for an informed decision, depending on the acceptability of risks.

2. When estimating the number of people attending an event, one may wish to estimate a likely maximum and minimum (eg end-points of a 95% confidence interval) to allow planning for facilities.

Sheet 02

Exercise 4: Logistic Regression

1)

$y \in \{0,1\}$, $p = p(y=1|x) = \sigma(w^T x) = \mu_1$ $y \sim Ber(p)$

$$\Rightarrow p(y=k|x) = \mu_k^k (1-\mu_k)^{1-k} \overset{*}{=} \sigma(w^T x)^k \sigma(-w^T x)^{1-k}$$

$* \; 1-\sigma(z) = \frac{\cancel{1}+e^{-z}-\cancel{1}}{1+e^{-z}} = \frac{1}{1+e^{z}} = \sigma(-z)$

Log likelihood:

$$\ell(w) = \sum_{i=1}^N \log(p(y_i|x_i)) = \sum_{i=1}^N \log(\sigma(w^T x_i)^{y_i} \sigma(-w^T x_i)^{1-y_i})$$
$$= \sum_{i=1}^N y_i \log(\sigma(w^T x_i)) + (1-y_i) \log(\sigma(-w^T x_i))$$

2) a)

Convexity of function ensures, that following the direction indicated by gradient descent you will find its global minimum, since any local minimum is the global minimum of a convex function.

To proof: $-\ell(w) = -\sum_{i=1}^N k \log(\sigma(w^T x_i)) + (1-k) \log(\sigma(-w^T x_i))$ is convex

Convexity: $f: D \in \mathbb{R} \to \mathbb{R}$ convex iff $f(tx_1 + (1-t)x_2) \leq t f(x_1) + (1-t) f(x_2)$ $\forall x_1, x_2 \in D$, $0 \leq t \leq 1$

Equivalently: f convex $\Leftrightarrow$ $f'' > 0$ $\forall x \in D$

i) First consider $-\log(\sigma(x))$:

$$\partial_x - \log(\sigma(x)) = \partial_x \log(1+e^{-x}) = \frac{-e^{-x}}{1+e^{-x}} = -\sigma(x)$$
$$\partial_x^2 - \log(\sigma(x)) = \partial_x \sigma(x) = \sigma(x)\sigma(-x) > 0 \quad \forall x \quad \Rightarrow -\log(\sigma(x)) \text{ is convex}$$

ii) Secondly consider: $-\log(\sigma(-x))$:

$$-\partial_x \log(\sigma(-x)) = \partial_x \log(1+e^{x}) = \frac{1}{1+e^{x}} e^{x} = \sigma(x)$$
$$\partial_x^2 - \log(\sigma(-x)) = \partial_x \sigma(x) = \sigma(x)\sigma(-x) > 0 \quad \forall x \quad \Rightarrow -\log(\sigma(-x)) \text{ is convex}$$

Since $-\ell(w)$ is a sum of convex functions of the form i) and ii) it is convex itself. ▨

b)

$$\mathcal{L}_{BCE}(w) = -N \ell(w)$$

$\Rightarrow$ maximizing log-likelihood $\Leftrightarrow$ minimizing average binary cross-entropy

3) a)

Suppose we have found ω s.t. $\omega^T x = 0$ defines the decision boundary between two linearly separable classes.

That would lead to a loss:

$$\mathcal{L}(\omega) = -\ell(\omega) = -\sum_{i=1}^{N} y_i \log(\sigma(\omega^T x_i)) + (1-y_i)\log(\sigma(-\omega^T x_i))$$

where the arguments of all σ-functions contributing to it will be positive, ie. $\sigma \in (0{,}5, 1]$, because the model performs "correct" classifications.

The only way for the model to optimize, i.e. minimize the loss, is now to have not only correct classifications, but also "confident" classifications, meaning $\sigma(\pm\omega^T x_i) \sim 1$ and thus $\log(\sigma(\pm\omega^T x_i)) \sim 0$. For this we need: $\|\omega^T x_i\| \longrightarrow \infty$, which the model will try to ensure by taking $|\omega|$ to infinity.

b)

This can be mitigated implementing a so-called L2-regularization in the Loss:

$$\tilde{\mathcal{L}}(\omega) = \mathcal{L}(\omega) + \lambda \|\omega\|^2$$

which penalizes high ω-magnitudes and prevents the divergence.