

ECE271A - Homework2

By Raghusrinivasan Venkatesan

Question 6

(a) From last week's observation we know that

$n_{\text{cheetah}} = 250$

$n_{\text{grass}} = 1053$

After solving the question 2 (using method of Langrange multipliers) I found out the MLE for prior probabilities is $\pi_{\text{hat}} = (n_{\text{samples}})/(\text{total_number_of_samples})$ [solution attached in appendix]

$\text{prior_cheetah} = 250/(250+1053) = 0.1919$

$\text{prior_grass} = 1053/(250+1053) = 0.8081$

(b) The maximum likelihood estimate for Gaussian distribution are

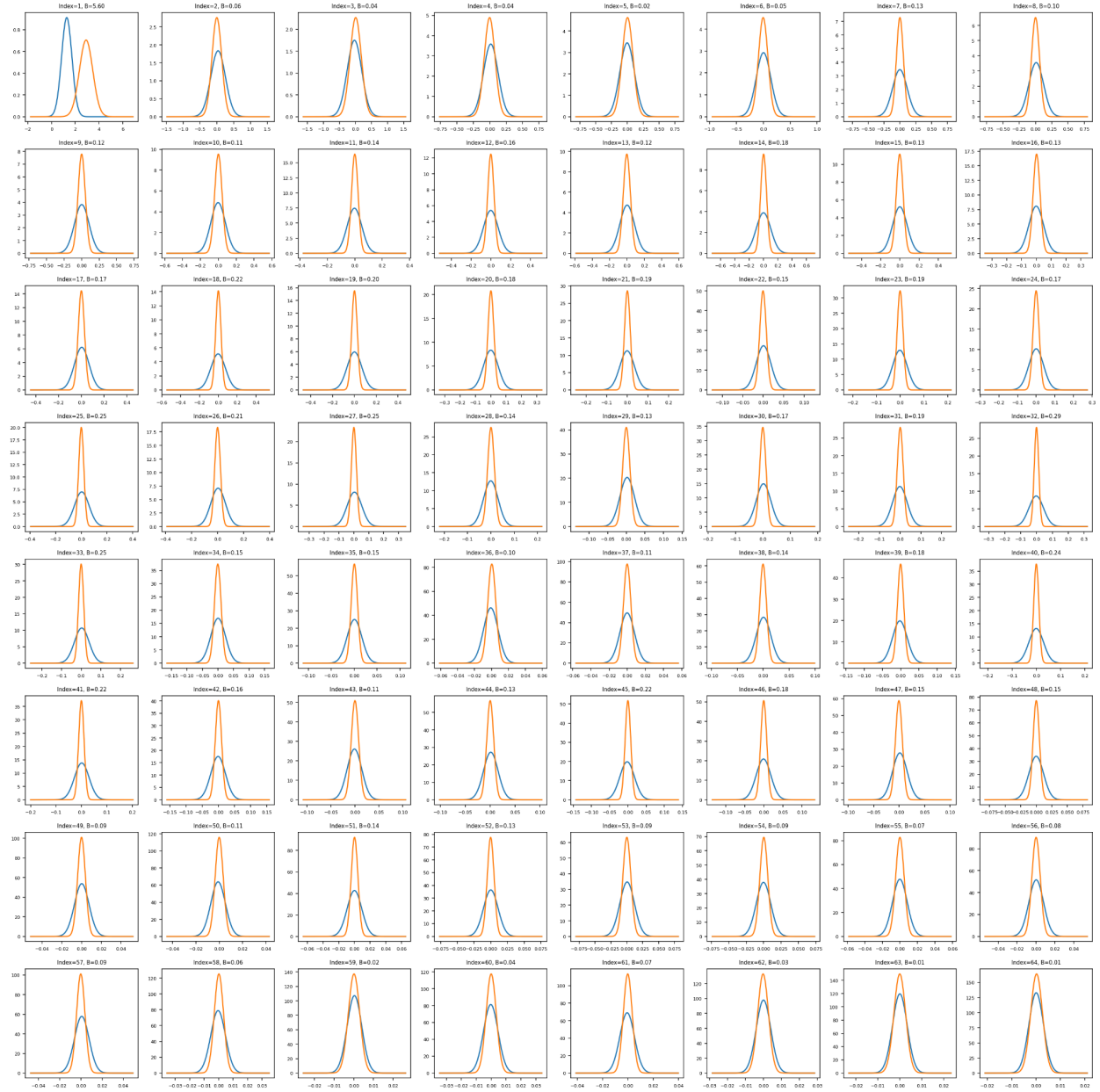
$$L(\mu, \sigma^2) = \prod_{i=1}^N f(x_i | \mu, \sigma^2) = \prod_{i=1}^N \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right)$$

$$L(\mu, \sigma^2) = (2\pi\sigma^2)^{-N/2} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^N (x_i - \mu)^2\right)$$

$$\ell(\mu, \sigma^2) = \log L(\mu, \sigma^2) = -\frac{N}{2} \log(2\pi) - \frac{N}{2} \log(\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^N (x_i - \mu)^2$$

The posterior probabilities can be calculated

The marginal distribution plots is as follows:



The best 8 indices were chosen based on the bhattacharya distance, the top 8 indices having the highest bhattacharya distance.

$$D_B = \frac{1}{4} \ln \left(\frac{1}{4} \left(\frac{\sigma_1^2}{\sigma_2^2} + \frac{\sigma_2^2}{\sigma_1^2} + 2 \right) \right) + \frac{1}{4} \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$$

After calculating the best and worst 8 indices with high bhattacharya distance comes out to be:

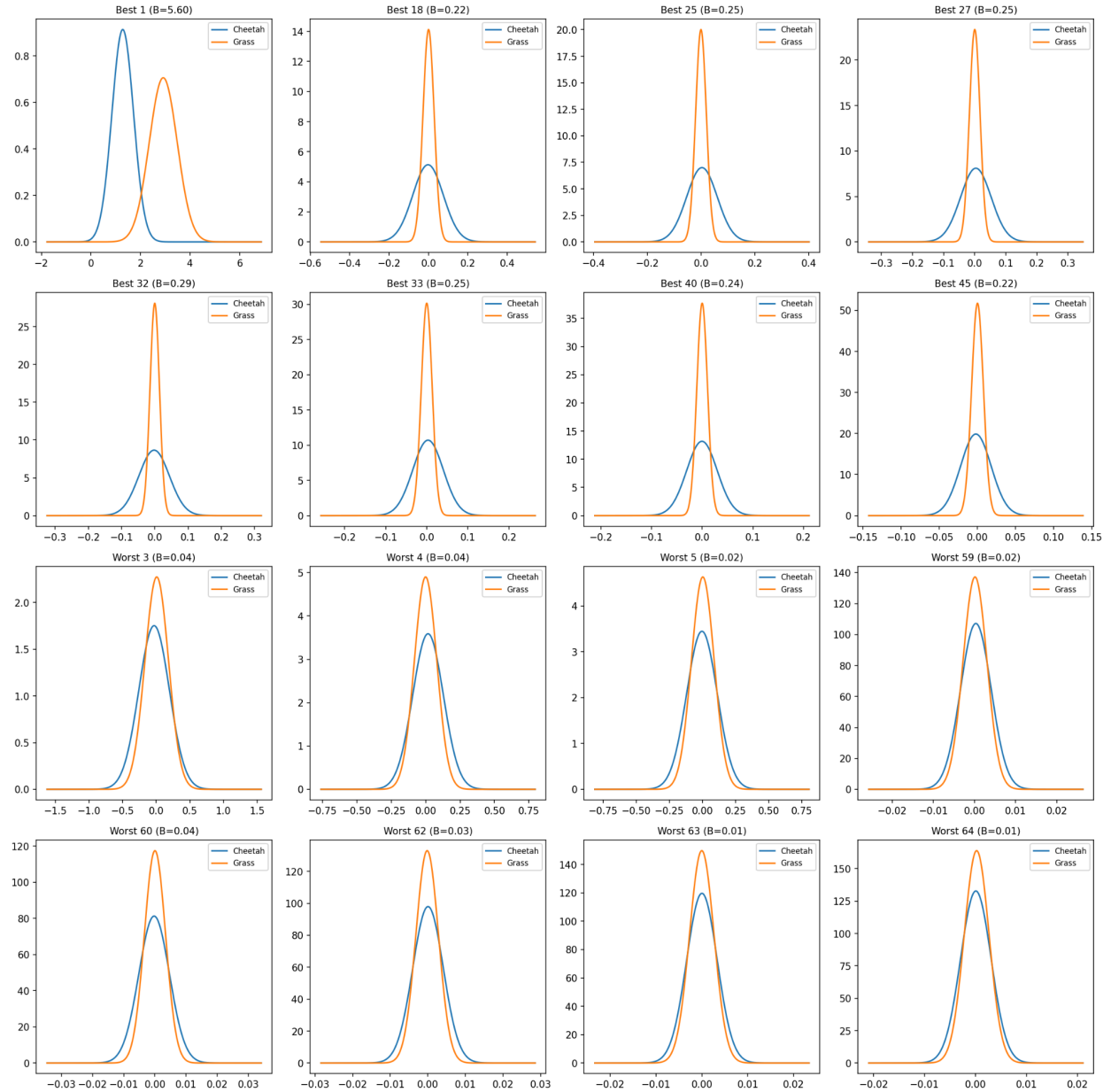
Best 8 indices: [1, 18, 25, 27, 32, 33, 40, 45]

With best distances being : [5.60314044, 0.22422459, 0.24552272, 0.25195321, 0.29200947
0.2462813, 0.23664597, 0.22317568]

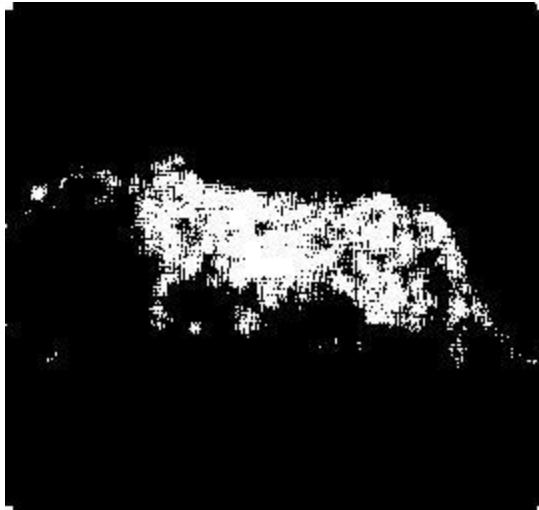
Worst 8 indices: [3, 4, 5, 59, 60, 62, 63, 64]

With worst distances being: [0.03646912, 0.04215876, 0.02369333, 0.01717606, 0.03547258,
0.02522119, 0.01255077, 0.01274599]

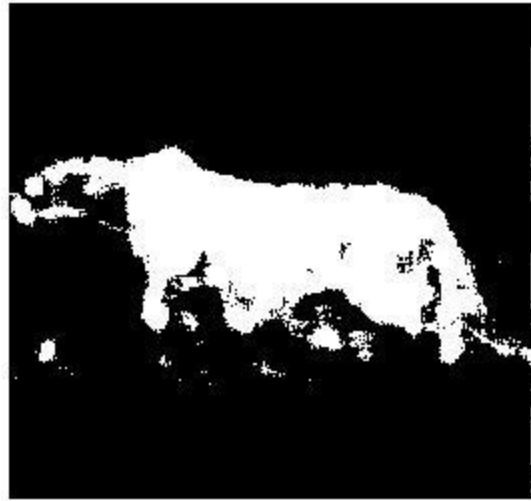
The Best and worst indices plots look like:



After using the Bayesian Decision Rule for classification of pixels in image:



8 Best Feature Prediction



64 Dimension feature Prediction

The error is calculated as:

$\text{Error}(\text{FG}) = [(\# \text{ of pixels misclassified as cheetah}) / (\# \text{ of pixels actually termed as cheetah in test set})] * \text{prior_cheetah}$

$\text{Error}(\text{BG}) = [(\# \text{ of pixels misclassified as grass}) / (\# \text{ of pixels actually termed as grass in test set})] * \text{prior_grass}$

$\text{Total error} = \text{Error}(\text{FG}) + \text{Error}(\text{BG})$

For 64D Gaussian the error turns out to be: $(0.032885 + 0.075293) = \mathbf{0.108178}$

For 8 best indices the error turns out to be: $(0.108178 + 0.099484) = \mathbf{0.099484}$

Observation:

The result shows the the error for 8 best indices feature prediction performs at par or slightly better than the 64 dimensional gaussian classifier, which infers that feature selection is significant for classification task and improves the performance of the classifier if the right features are selected (I tried with Fisher's discriminant ratio and the feature selection was not good in turn giving higher error than Bhattacharya distance).

Why is this happening is the next question in our minds?

One potential reason is that there might be some misleading features in the 64 dimension Gaussian predictor which would be giving us wrong predictions and if the number of such features is high then it could have a high impact on our classification accuracy. Hence, feature selection turns out to be an important step for classification. Good features don't tend to have similar probability distribution unlike misleading features so they can easily distinguish if the pixel corresponds to cheetah or grass, hence using the set of good features in most cases can provide better results than using all the features.

Appendix

Code:

```
# -*- coding: utf-8 -*-
"""hw2_ECE271A_Bhattacharya_Distance.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1U1f83o-u2qsNqI0tGEcT7ExIYoCfRD3Z
"""

import numpy as np
import matplotlib.pyplot as plt
import scipy
from scipy.io import loadmat
from scipy.fftpack import dct
from PIL import Image
import warnings
warnings.filterwarnings('ignore')

def zigzag(matrix):
    zigzag_pattern = [
        [0, 1, 5, 6, 14, 15, 27, 28],
        [2, 4, 7, 13, 16, 26, 29, 42],
        [3, 8, 12, 17, 25, 30, 41, 43],
        [9, 11, 18, 24, 31, 40, 44, 53],
        [10, 19, 23, 32, 39, 45, 52, 54],
        [20, 22, 33, 38, 46, 51, 55, 60],
        [21, 34, 37, 47, 50, 56, 59, 61],
        [35, 36, 48, 49, 57, 58, 62, 63]
    ]

    result = np.zeros(64)
    for i in range(8):
        for j in range(8):
            result[zigzag_pattern[i][j]] = matrix[i, j]
    return result
```

```

def dct2(block):
    return dct(dct(block.T, norm='ortho').T, norm='ortho')

def bhattacharyya_distance(mean1, var1, mean2, var2):
    """
    Calculate Bhattacharyya Distance for univariate Gaussian distributions

    Formula for univariate Gaussians:
     $DB = 0.25 * \ln(0.25 * (\sigma_1^2/\sigma_2^2 + \sigma_2^2/\sigma_1^2 + 2)) + 0.25 * (\mu_1 - \mu_2)^2 * (1/\sigma_1^2 + 1/\sigma_2^2)$ 

    Higher distance = better class separation
    Range: [0,  $\infty$ )
    """
    var1 = np.where(var1 == 0, 1e-10, var1)
    var2 = np.where(var2 == 0, 1e-10, var2)

    term1 = 0.25 * np.log(0.25 * (var1/var2 + var2/var1 + 2))

    term2 = 0.25 * (mean1 - mean2) ** 2 * (1/var1 + 1/var2)

    return term1 + term2

train_set = scipy.io.loadmat('TrainingSamplesDCT_8_new.mat')

background_data = train_set['TrainsampleDCT_BG']
cheetah_data = train_set['TrainsampleDCT_FG']

c_grass = background_data.shape[0]
c_cheetah = cheetah_data.shape[0]
n = c_grass + c_cheetah

print(f"Number of background samples: {c_grass}")
print(f"Number of cheetah samples: {c_cheetah}")
print(f"Feature dimension: {background_data.shape[1]}")

prior_cheetah = c_cheetah / n
prior_background = c_grass / n

print(f"Prior probability of cheetah: {prior_cheetah:.4f}")
print(f"Prior probability of background: {prior_background:.4f}")

```

```

print("\n" + "=" * 60)
print("Problem B: Loading training data and computing statistics")
print("=" * 60)

train_set = loadmat('TrainingSamplesDCT_8_new.mat')
FGmat = train_set['TrainsampleDCT_FG']
BGmat = train_set['TrainsampleDCT_BG']

print(f"Foreground samples shape: {FGmat.shape}")
print(f"Background samples shape: {BGmat.shape}")

meanFG = np.mean(FGmat, axis=0)
covFG = np.cov(FGmat, rowvar=False)
varFG = np.var(FGmat, axis=0)

meanBG = np.mean(BGmat, axis=0)
covBG = np.cov(BGmat, rowvar=False)
varBG = np.var(BGmat, axis=0)

mleFG = np.zeros(c_cheetah)
for i in range(c_cheetah):
    tmp = 0
    for j in range(64):
        tmp += ((FGmat[i, j] - meanFG[j]) / np.sqrt(varFG[j])) ** 2
    mleFG[i] = np.exp(-32 * np.log(2 * np.pi) - np.sum(np.log(np.sqrt(varFG))) - 0.5 *
tmp)

mleBG = np.zeros(c_grass)
for i in range(c_grass):
    tmp = 0
    for j in range(64):
        tmp += ((BGmat[i, j] - meanBG[j]) / np.sqrt(varBG[j])) ** 2
    mleBG[i] = np.exp(-32 * np.log(2 * np.pi) - np.sum(np.log(np.sqrt(varBG))) - 0.5 *
tmp)

print("\nCalculating Bhattacharyya Distance for feature selection...")
bhattacharyya_distances = np.zeros(64)

for i in range(64):
    ave_FG = np.mean(FGmat[:, i])
    variance_FG = np.var(FGmat[:, i])
    ave_BG = np.mean(BGmat[:, i])

```

```

    variance_BG = np.var(BGmat[:, i])

    bhattacharyya_distances[i] = bhattacharyya_distance(ave_FG, variance_FG, ave_BG,
variance_BG)

print(f"Bhattacharyya distances calculated for all 64 features")
print(f"Bhattacharyya distance range: [{bhattacharyya_distances.min():.4f},
{bhattacharyya_distances.max():.4f}]")

print("\nGenerating marginal plots for all 64 dimensions...")
fig = plt.figure(figsize=(20, 20))

for i in range(64):
    plt.subplot(8, 8, i + 1)

    ave_FG = np.mean(FGmat[:, i])
    variance_FG = np.var(FGmat[:, i])
    sigma_FG = np.sqrt(variance_FG)

    ave_BG = np.mean(BGmat[:, i])
    variance_BG = np.var(BGmat[:, i])
    sigma_BG = np.sqrt(variance_BG)

    xFG = np.linspace(ave_FG - 7 * sigma_FG, ave_FG + 7 * sigma_FG, 350)
    xBG = np.linspace(ave_BG - 7 * sigma_BG, ave_BG + 7 * sigma_BG, 350)
    x = np.sort(np.concatenate([xFG, xBG]))

    y_cheetah = (1 / (sigma_FG * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x - ave_FG) /
sigma_FG) ** 2)
    y_grass = (1 / (sigma_BG * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x - ave_BG) /
sigma_BG) ** 2)

    plt.plot(x, y_cheetah, label='Cheetah')
    plt.plot(x, y_grass, label='Grass')
    plt.title(f'Index={i + 1}, B={bhattacharyya_distances[i]:.2f}', fontsize=7)
    plt.tick_params(labelsize=6)

plt.tight_layout()
plt.savefig('64_plots_bhattacharyya.png', dpi=150, bbox_inches='tight')
print("Saved: 64_plots_bhattacharyya.png")
plt.close()

```

```

BhattacharyyaIdx = np.argsort(bhattacharyya_distances)
worstidx = np.sort(BhattacharyyaIdx[:8])
bestidx = np.sort(BhattacharyyaIdx[56:64])

print(f"\nFeature Selection Results (Bhattacharyya Distance):")
print(f"Best 8 features (highest Bhattacharyya distances): {bestidx + 1}")
print(f"  Bhattacharyya distances: {bhattacharyya_distances[bestidx]}")
print(f"Worst 8 features (lowest Bhattacharyya distances): {worstidx + 1}")
print(f"  Bhattacharyya distances: {bhattacharyya_distances[worstidx]}")

fig = plt.figure(figsize=(16, 16))

for i in range(8):
    idx_w = worstidx[i]
    idx_b = bestidx[i]

    plt.subplot(4, 4, i + 1)
    ave_FG = np.mean(FGmat[:, idx_b])
    variance_FG = np.var(FGmat[:, idx_b])
    sigma_FG = np.sqrt(variance_FG)
    ave_BG = np.mean(BGmat[:, idx_b])
    variance_BG = np.var(BGmat[:, idx_b])
    sigma_BG = np.sqrt(variance_BG)

    xFG = np.linspace(ave_FG - 7 * sigma_FG, ave_FG + 7 * sigma_FG, 350)
    xBG = np.linspace(ave_BG - 7 * sigma_BG, ave_BG + 7 * sigma_BG, 350)
    x = np.sort(np.concatenate([xFG, xBG]))

    y_cheetah = (1 / (sigma_FG * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x - ave_FG) /
sigma_FG) ** 2)
    y_grass = (1 / (sigma_BG * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x - ave_BG) /
sigma_BG) ** 2)

    plt.plot(x, y_cheetah, label='Cheetah')
    plt.plot(x, y_grass, label='Grass')
    plt.title(f'Best {idx_b + 1} (B={bhattacharyya_distances[idx_b]:.2f})',
fontsize=10)
    plt.legend(fontsize=8)

    plt.subplot(4, 4, i + 9)
    ave_FG = np.mean(FGmat[:, idx_w])
    variance_FG = np.var(FGmat[:, idx_w])

```

```

sigma_FG = np.sqrt(variance_FG)
ave_BG = np.mean(BGmat[:, idx_w])
variance_BG = np.var(BGmat[:, idx_w])
sigma_BG = np.sqrt(variance_BG)

xFG = np.linspace(ave_FG - 7 * sigma_FG, ave_FG + 7 * sigma_FG, 350)
xBG = np.linspace(ave_BG - 7 * sigma_BG, ave_BG + 7 * sigma_BG, 350)
x = np.sort(np.concatenate([xFG, xBG]))

y_cheetah = (1 / (sigma_FG * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x - ave_FG) /
sigma_FG) ** 2)
y_grass = (1 / (sigma_BG * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x - ave_BG) /
sigma_BG) ** 2)

plt.plot(x, y_cheetah, label='Cheetah')
plt.plot(x, y_grass, label='Grass')
plt.title(f'Worst {idx_w + 1} (B={bhattacharyya_distances[idx_w]:.2f})',
fontsize=10)
plt.legend(fontsize=8)

plt.tight_layout()
plt.savefig('BestandWorst_plots_bhattacharyya.png', dpi=150, bbox_inches='tight')
print("Saved: BestandWorst_plots_bhattacharyya.png")
plt.close()

print("\n" + "=" * 60)
print("Problem C(i): Classification using 64D Gaussian")
print("=" * 60)

img = np.array(Image.open('cheetah.bmp').convert('L'), dtype=np.float64) / 255.0

img = np.pad(img, ((4, 3), (4, 3)), mode='constant', constant_values=0)

m, n = img.shape
Blocks = np.ones((m - 7, n - 7))

mean_FG_full = meanFG
mean_BG_full = meanBG
inv_covFG = np.linalg.inv(covFG)
inv_covBG = np.linalg.inv(covBG)
DcovFG = np.linalg.det(covFG)
DcovBG = np.linalg.det(covBG)

```

```

print("Processing image blocks...")
for i in range(m - 7):
    if (i + 1) % 50 == 0:
        print(f"  Progress: {i + 1}/{m - 7} rows")
    for j in range(n - 7):
        block = img[i:i + 8, j:j + 8]
        DCT = dct2(block)
        feature = zigzag(DCT)
        g_cheetah = (feature @ inv_covFG @ feature.T -
                     2 * feature @ inv_covFG @ mean_FG_full.T +
                     mean_FG_full @ inv_covFG @ mean_FG_full.T +
                     np.log(DcovFG) - 2 * np.log(prior_cheetah))

        g_grass = (feature @ inv_covBG @ feature.T -
                   2 * feature @ inv_covBG @ mean_BG_full.T +
                   mean_BG_full @ inv_covBG @ mean_BG_full.T +
                   np.log(DcovBG) - 2 * np.log(prior_background))

        if g_cheetah >= g_grass:
            Blocks[i, j] = 0

Image.fromarray((Blocks *
255).astype(np.uint8)).save('prediction_64d_bhattacharyya.jpg')
print("Saved: prediction_64d_bhattacharyya.jpg")

ground_truth = np.array(Image.open('cheetah_mask.bmp'), dtype=np.float64) / 255.0
prediction = Blocks

x, y = ground_truth.shape
count1 = 0
count2 = 0
count_cheetah_truth = 0
count_grass_truth = 0

for i in range(x):
    for j in range(y):
        if prediction[i, j] > ground_truth[i, j]:
            count2 += 1
            count_grass_truth += 1
        elif prediction[i, j] < ground_truth[i, j]:
            count1 += 1

```

```

        count_cheetah_truth += 1
    elif ground_truth[i, j] > 0:
        count_cheetah_truth += 1
    else:
        count_grass_truth += 1

error1_64 = (count1 / count_cheetah_truth) * prior_cheetah
error2_64 = (count2 / count_grass_truth) * prior_background
total_error_64 = error1_64 + error2_64

print(f"\n64D Results:")
print(f"  Error 1 (miss cheetah): {error1_64:.6f}")
print(f"  Error 2 (false alarm): {error2_64:.6f}")
print(f"  Total error: {total_error_64:.6f}")

print("\n" + "=" * 60)
print("Problem C(ii): Classification using 8 best features")
print(f"                (Selected by Bhattacharyya Distance)")
print("=" * 60)

img = np.array(Image.open('cheetah.bmp').convert('L'), dtype=np.float64) / 255.0
img = np.pad(img, ((4, 3), (4, 3)), mode='constant', constant_values=0)

m, n = img.shape
Blocks = np.ones((m - 7, n - 7))

mean_FG_best = np.mean(FGmat[:, bestidx], axis=0)
mean_BG_best = np.mean(BGmat[:, bestidx], axis=0)
cov_cheetah = np.cov(FGmat[:, bestidx], rowvar=False)
cov_grass = np.cov(BGmat[:, bestidx], rowvar=False)
inv_covFG_best = np.linalg.inv(cov_cheetah)
inv_covBG_best = np.linalg.inv(cov_grass)
DcovFG_best = np.linalg.det(cov_cheetah)
DcovBG_best = np.linalg.det(cov_grass)

print("Processing image blocks...")
for i in range(m - 7):
    if (i + 1) % 50 == 0:
        print(f"  Progress: {i + 1}/{m - 7} rows")
    for j in range(n - 7):
        block = img[i:i + 8, j:j + 8]
        DCT = dct2(block)

```

```

zigzag_full = zigzag(DCT)
feature = zigzag_full[bestidx]
g_cheetah = (feature @ inv_covFG_best @ feature.T -
              2 * feature @ inv_covFG_best @ mean_FG_best.T +
              mean_FG_best @ inv_covFG_best @ mean_FG_best.T +
              np.log(DcovFG_best) - 2 * np.log(prior_cheetah))

g_grass = (feature @ inv_covBG_best @ feature.T -
            2 * feature @ inv_covBG_best @ mean_BG_best.T +
            mean_BG_best @ inv_covBG_best @ mean_BG_best.T +
            np.log(DcovBG_best) - 2 * np.log(prior_background))

if g_cheetah >= g_grass:
    Blocks[i, j] = 0

Image.fromarray((Blocks *
255).astype(np.uint8)).save('prediction_8d_bhattacharyya.jpg')
print("Saved: prediction_8d_bhattacharyya.jpg")

prediction = Blocks
count1 = 0
count2 = 0
count_cheetah_truth = 0
count_grass_truth = 0

for i in range(x):
    for j in range(y):
        if prediction[i, j] > ground_truth[i, j]:
            count2 += 1
            count_grass_truth += 1
        elif prediction[i, j] < ground_truth[i, j]:
            count1 += 1
            count_cheetah_truth += 1
        elif ground_truth[i, j] > 0:
            count_cheetah_truth += 1
        else:
            count_grass_truth += 1

error1_8 = (count1 / count_cheetah_truth) * prior_cheetah
error2_8 = (count2 / count_grass_truth) * prior_background
total_error_8 = error1_8 + error2_8

```

```
print(f"\n8D Results (Bhattacharyya-selected features):")
print(f"  Error 1 (miss cheetah): {error1_8:.6f}")
print(f"  Error 2 (false alarm): {error2_8:.6f}")
print(f"  Total error: {total_error_8:.6f}")

print("\n" + "=" * 60)
print("SUMMARY (Using Bhattacharyya Distance)")
print("=" * 60)
print(f"Feature Selection Method: Bhattacharyya Distance")
print(f"Best 8 features: {bestidx + 1}")
print(f"64D Gaussian - Total Error: {total_error_64:.6f}")
print(f"8D Gaussian - Total Error: {total_error_8:.6f}")
print("=" * 60)
```

Derivation (Ques 2):

HW2 problem 2

(multinomial distribution)

$$P(c_1, c_2, c_3, \dots, c_N) = \frac{n!}{\prod_{k=1}^N c_k!} \prod_{j=1}^N \pi_j^{c_j}$$

$$\text{likelihood function} = L(\pi_1, \pi_2, \pi_3, \dots, \pi_N) = \frac{n!}{\prod_{k=1}^N c_k!} \prod_{j=1}^N \pi_j^{c_j}$$

taking log-likelihood

$$\log(L(\pi_1, \pi_2, \dots, \pi_N)) = \log(n!) - \sum_{k=1}^N \log(c_k!) + \sum_{j=1}^N c_j \log(\pi_j)$$

can be ignored
as they are constant

$$\log(L(\pi_1, \pi_2, \dots, \pi_N)) = \sum_{j=1}^N c_j \log(\pi_j)$$

$$\text{constraint} \left[\sum_{k=1}^N \pi_k = 1 \right]$$

Now, using method of Lagrangian multipliers

$$\mathcal{L}(\pi_1, \pi_2, \dots, \pi_N, \lambda) = \sum_{j=1}^N c_j \log(\pi_j) + \lambda \left(1 - \sum_{j=1}^N \pi_j \right)$$

Now, taking partial derivatives

$$\frac{\partial \mathcal{L}}{\partial \pi_k} = \frac{c_k}{\pi_k} - \lambda = 0$$

$$\boxed{\pi_k = \frac{c_k}{\lambda}}$$

Now solving for λ .

$$\sum_{k=1}^N \pi_k = 1 \Rightarrow \sum_{k=1}^N \frac{C_k}{\lambda} = 1 \quad \text{sum of } \pi_k$$

$$\prod_{k=1}^N \pi_k = 1 \quad \left(\sum_{k=1}^N C_k = 1 \right)$$

$$\prod_{k=1}^N \pi_k = 1 \Rightarrow \left[\lambda = n \right]$$

$$\hat{\pi}_k = \frac{C_k}{n} \quad \text{for } k=1, 2, 3, \dots, N$$

$$(1, 0, \dots, 0) \cdot \hat{\pi} = (1, 0, \dots, 0) \cdot \frac{1}{n} (C_1, C_2, \dots, C_N) = \frac{C_1}{n}$$

example: let $\pi = (1, 0, \dots, 0)$

then $\pi \cdot \hat{\pi} = \frac{C_1}{n}$

$$(\pi) \cdot \hat{\pi} = (1, 0, \dots, 0) \cdot \frac{1}{n} (C_1, C_2, \dots, C_N)$$

Now we find the value of π

$$\left[1 = \sum_{k=1}^N \pi_k \right] \quad \text{constraint}$$

$$(\pi \cdot \hat{\pi} - 1) \lambda + (\pi \cdot \hat{\pi}) \lambda = (\lambda, \lambda, \dots, \lambda)$$

Now we find the value of λ

$$\left(\frac{C_k}{\lambda} = \pi_k \right)$$

$$0 = \lambda - \frac{C_k}{\pi_k} = \frac{C_k}{\pi_k} - \lambda$$

Now we find the value of λ

