# ECE 271A: Statistical Learning I Quiz Report

JunPyung Kim
PID: A69044427
University of California, San Diego

October 27, 2025

## 1 Quiz 2: Gaussian Bayesian Classifier for Image Segmentation

### 1.1 Objective

The goal of this assignment is to extend the previous Bayesian classifier by modeling the class-conditional densities, $P_{X|Y}(x|c)$, as multivariate Gaussian distributions. We will compare the performance of a full 64-dimensional Gaussian model against a reduced 8-dimensional model built using the most discriminative features.

### 1.2 Methodology and Results

#### 1.2.1 Part (a): Prior Probabilities

The prior probabilities, $P(Y = \text{cheetah})$ and $P(Y = \text{grass})$, were re-evaluated using the new `TrainingSamplesDCT_8_new.mat` data. The Maximum Likelihood Estimate (MLE) for the parameter $p_c$ of a categorical distribution is given by:

$$\hat{p}_c = P(Y = c) = \frac{N_c}{N_{\text{total}}}$$

where $N_c$ is the number of samples for class $c$. Based on the 250 foreground (cheetah) and 1053 background (grass) samples:

- $P(Y = \text{cheetah}) = \frac{250}{250+1053} = \frac{250}{1303} \approx 0.1919$
- $P(Y = \text{grass}) = \frac{1053}{250+1053} = \frac{1053}{1303} \approx 0.8081$

These results are identical to the priors computed in Quiz 1.

#### 1.2.2 Part (b): Class-Conditional Parameters and Feature Selection

The class-conditional densities were modeled as 64-dimensional Gaussian distributions, $P_{X|Y}(x|c) \sim \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$. The MLE parameters for the mean vector $\boldsymbol{\mu}_c$ and covariance matrix $\boldsymbol{\Sigma}_c$ were computed from the training data. The plot for all 64 marginal densities is shown in Figure 1.

Feature selection was performed quantitatively using the **Symmetric Kullback-Leibler (KL) Divergence**. This metric measures the "distance" between the 1D marginal distributions $P(X_k|cheetah)$ and $P(X_k|grass)$ for each feature. The 8 features with the highest KL divergence (most separable) and the 8 with the lowest (least separable) were identified.

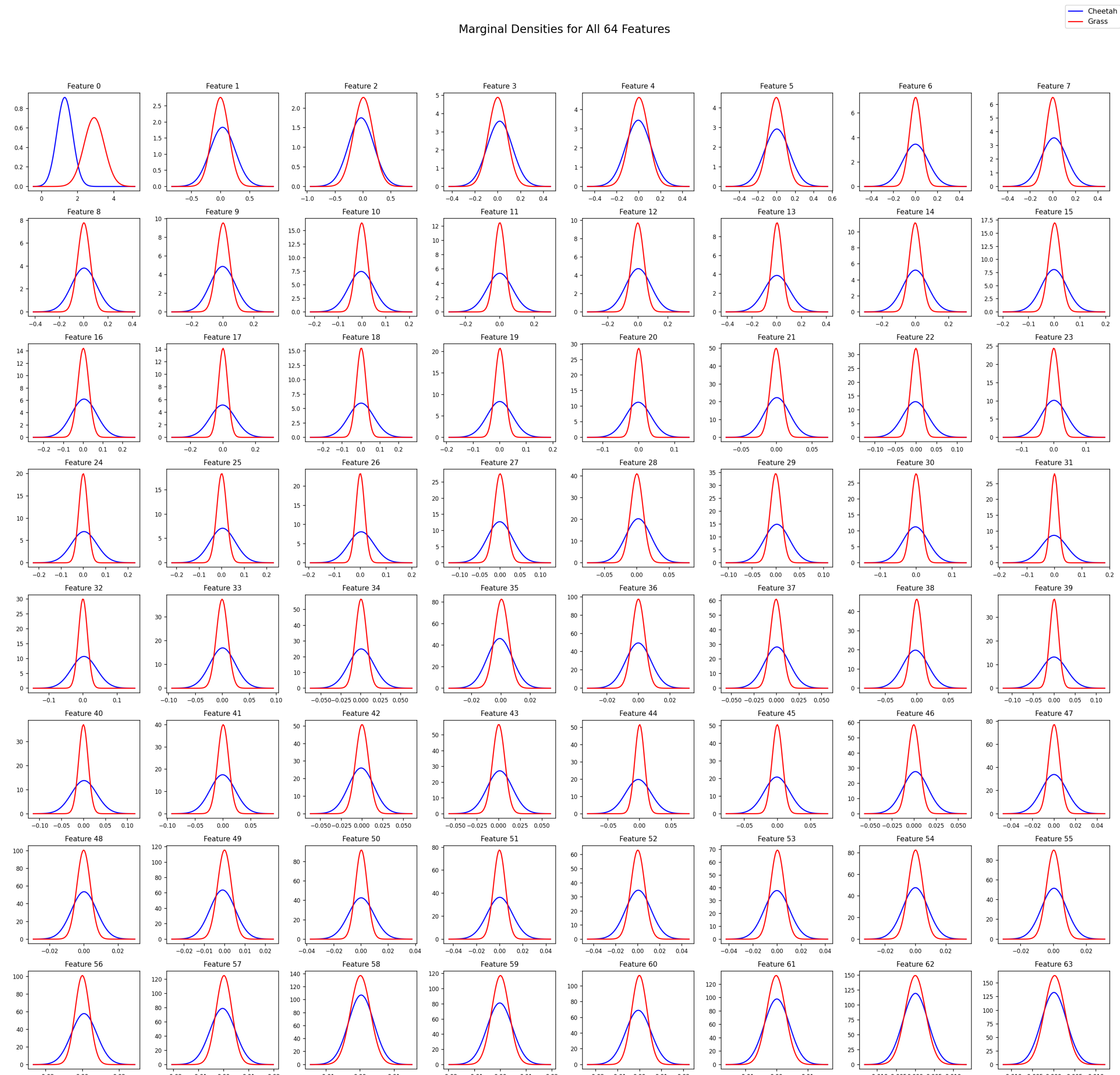


Figure 1: Marginal densities $P(X_k|Y)$ for all 64 DCT features.

- **Best 8 Features (by KL Div):** [0, 31, 26, 24, 39, 32, 17, 40]
- **Worst 8 Features (by KL Div):** [63, 62, 58, 2, 4, 61, 3, 59]

The marginal densities for these selected features are plotted in Figure 2.

### 1.2.3 Part (c): Image Segmentation and Performance

The `cheetah.bmp` image was classified using a sliding 8x8 window. At each pixel, the 64-D DCT vector for the corresponding block was extracted and classified using the Bayes Decision Rule for

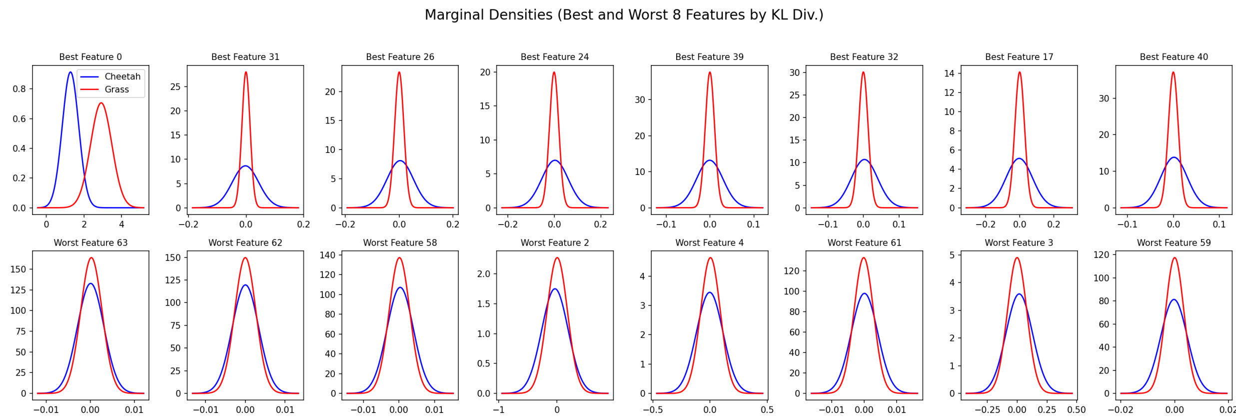


Figure 2: Marginal densities $P(X_k|Y)$ for the 8 best (top) and 8 worst (bottom) features, as selected by Symmetric KL Divergence.

minimum error:

$$\hat{Y} = \arg\max_{Y\in\{\text{cheetah, grass}\}} \left[\log P_{X|Y}(x|Y) + \log P(Y)\right]$$

This classification was performed twice:

1. Using the full 64-dimensional Gaussian models.
2. Using 8-dimensional Gaussian models built from the best 8 features.

The resulting probability of error for each classifier, computed by comparing the output mask to the ground truth, was:

- **64-D Classifier Error:** 0.1450 (14.50%)
- **8-D Classifier Error:** 0.0748 (7.48%)

### 1.2.4 Discussion: Explaining the Results

As shown by the error rates and the segmentation masks (Figures 3 and 4), the **8-dimensional classifier performs significantly better** than the 64-dimensional one.

This result is a classic example of the **"Curse of Dimensionality."** With a limited number of training samples (250 for cheetah, 1053 for grass), accurately estimating the parameters for a 64-dimensional Gaussian distribution is extremely difficult. The 64x64 covariance matrix has $\frac{64\times 65}{2} = 2080$ unique parameters. Estimating this many parameters from only 250 foreground samples leads to a model that is highly overfit to the training data and does not generalize well.

The 8-dimensional model, in contrast, requires estimating an 8x8 covariance matrix (36 parameters). This model is much simpler, its parameters can be estimated more robustly from the available data, and it consequently generalizes better to the unseen test image, resulting in a lower probability of error.

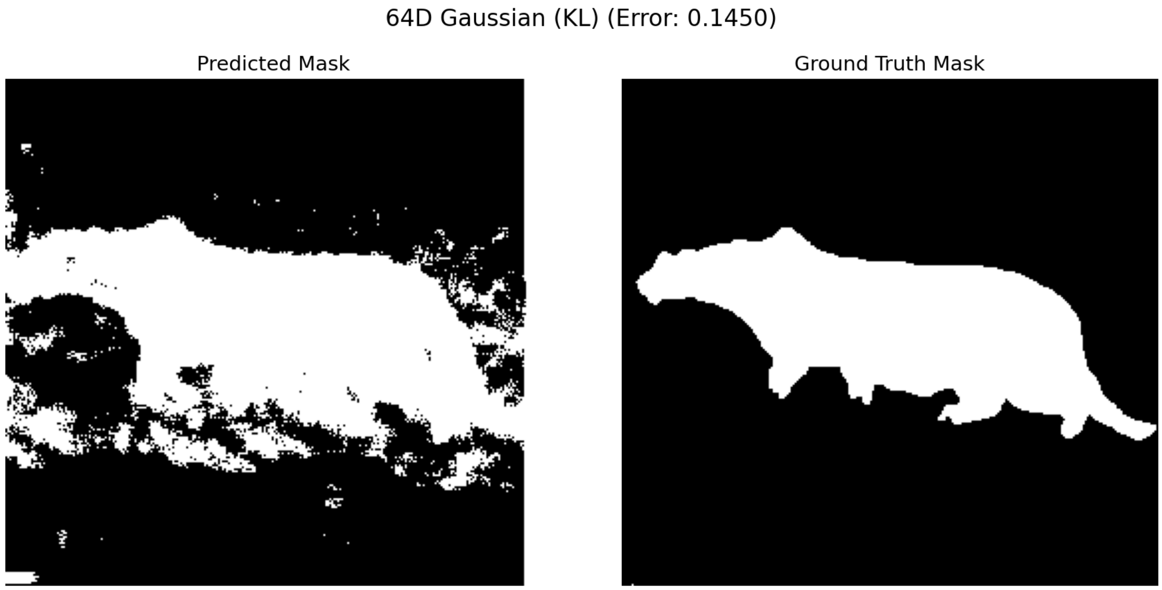


Figure 3: Segmentation using the full 64-D Gaussian model. Left: Predicted Mask. Right: Ground Truth. (Error: 14.50%)

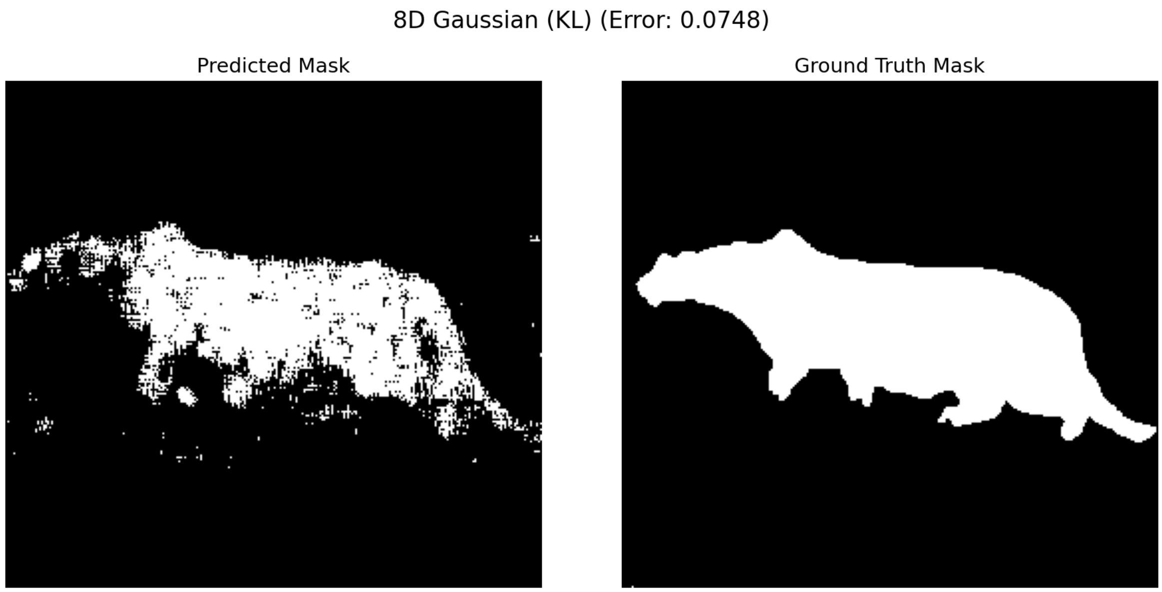


Figure 4: Segmentation using the 8-D Gaussian model (best features). Left: Predicted Mask. Right: Ground Truth. This model achieves a lower error rate. (Error: 7.48%)

### 1.3 Appendix: Source Code

The Python source code used for this assignment is attached below.

```python
import numpy as np
import scipy.io
import scipy.stats
```

```python
import imageio.v3 as imageio
import matplotlib
import os
import warnings
from scipy.fftpack import dctn
from tqdm import tqdm

try:
    matplotlib.use('Agg')
except ImportError:
    pass

import matplotlib.pyplot as plt

warnings.filterwarnings('ignore')
np.random.seed(42)

BLOCK_SIZE = 8
DATA_FILE = 'data/TrainingSamplesDCT_8_new.mat'
ZIG_ZAG_FILE = 'data/Zig-Zag Pattern.txt'
IMAGE_FILE = 'data/cheetah.bmp'
MASK_FILE = 'data/cheetah_mask.bmp'
OUTPUT_DIR = 'hw2/output/'


def load_zig_zag_map(filepath):
    """Loads the zig-zag scan pattern."""
    zig_zag_pattern = np.loadtxt(filepath, dtype=int)
    return np.argsort(zig_zag_pattern.flatten())


def dct2(block):
    """Compute 2D DCT of an 8x8 block."""
    return dctn(block, type=2, norm='ortho')


def compute_mle_parameters(data):
    """
    Compute MLE for mean and covariance matrix (ddof=0).
    """
    mean = np.mean(data, axis=0)
    cov = np.cov(data, rowvar=False, ddof=0)
    reg = np.eye(cov.shape[0]) * 1e-6
    cov_reg = cov + reg
    return mean, cov_reg


def compute_kl_divergence(p, q):
    """Compute symmetric KL divergence between two 1D Gaussian distributions."""
    mu_p, var_p = p
    mu_q, var_q = q
    epsilon = 1e-10
    var_p = max(var_p, epsilon)
    var_q = max(var_q, epsilon)

    kl_pq = 0.5 * (np.log(var_q / var_p) + (var_p + (mu_p - mu_q) ** 2) / var_q -
    1)
    kl_qp = 0.5 * (np.log(var_p / var_q) + (var_q + (mu_q - mu_p) ** 2) / var_p -
    1)
```

```python
    return kl_pq + kl_qp


def select_features_kl(fg_data, bg_data):
    """Select best and worst 8 features based on symmetric KL divergence."""
    kl_divergences = []
    for i in range(64):
        mean_fg = np.mean(fg_data[:, i])
        var_fg = np.var(fg_data[:, i], ddof=0)
        mean_bg = np.mean(bg_data[:, i])
        var_bg = np.var(bg_data[:, i], ddof=0)
        kl = compute_kl_divergence((mean_fg, var_fg), (mean_bg, var_bg))
        kl_divergences.append(kl)

    sorted_indices = np.argsort(kl_divergences)
    worst_8_indices = sorted_indices[:8]
    best_8_indices = sorted_indices[-8:][::-1]  # Top 8
    return best_8_indices, worst_8_indices


def plot_best_worst_features(fg_data, bg_data, best_indices, worst_indices, output_path):
    """Plot marginals for best and worst 8 features."""
    fig, axes = plt.subplots(2, 8, figsize=(20, 7))
    fig.suptitle("Marginal Densities (Best and Worst 8 Features by KL Div.)", fontsize=16)

    for i, idx in enumerate(best_indices):
        ax = axes[0, i]
        mean_fg = np.mean(fg_data[:, idx])
        std_fg = np.std(fg_data[:, idx], ddof=0)
        mean_bg = np.mean(bg_data[:, idx])
        std_bg = np.std(bg_data[:, idx], ddof=0)

        x_min = min(mean_fg - 4 * std_fg, mean_bg - 4 * std_bg)
        x_max = max(mean_fg + 4 * std_fg, mean_bg + 4 * std_bg)
        x = np.linspace(x_min, x_max, 200)

        ax.plot(x, scipy.stats.norm.pdf(x, mean_fg, std_fg), 'b-', label='Cheetah')
        ax.plot(x, scipy.stats.norm.pdf(x, mean_bg, std_bg), 'r-', label='Grass')
        ax.set_title(f'Best Feature {idx}', fontsize=10)
        if i == 0: ax.legend()

    for i, idx in enumerate(worst_indices):
        ax = axes[1, i]
        mean_fg = np.mean(fg_data[:, idx])
        std_fg = np.std(fg_data[:, idx], ddof=0)
        mean_bg = np.mean(bg_data[:, idx])
        std_bg = np.std(bg_data[:, idx], ddof=0)

        x_min = min(mean_fg - 4 * std_fg, mean_bg - 4 * std_bg)
        x_max = max(mean_fg + 4 * std_fg, mean_bg + 4 * std_bg)
        x = np.linspace(x_min, x_max, 200)

        ax.plot(x, scipy.stats.norm.pdf(x, mean_fg, std_fg), 'b-')
        ax.plot(x, scipy.stats.norm.pdf(x, mean_bg, std_bg), 'r-')
        ax.set_title(f'Worst Feature {idx}', fontsize=10)

```

```python
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.savefig(output_path, dpi=150)
    plt.close(fig)


def plot_all_features(fg_data, bg_data, output_path):
    """Plot marginals for all 64 features."""
    fig, axes = plt.subplots(8, 8, figsize=(22, 22))
    fig.suptitle("Marginal Densities for All 64 Features", fontsize=16)

    for i in range(64):
        ax = axes[i // 8, i % 8]

        mean_fg = np.mean(fg_data[:, i])
        std_fg = np.std(fg_data[:, i], ddof=0)
        mean_bg = np.mean(bg_data[:, i])
        std_bg = np.std(bg_data[:, i], ddof=0)

        std_fg = max(std_fg, 1e-6)
        std_bg = max(std_bg, 1e-6)

        x_min = min(mean_fg - 4 * std_fg, mean_bg - 4 * std_bg)
        x_max = max(mean_fg + 4 * std_fg, mean_bg + 4 * std_bg)
        x = np.linspace(x_min, x_max, 100)

        ax.plot(x, scipy.stats.norm.pdf(x, mean_fg, std_fg), 'b-', label='Cheetah'
     if i == 0 else "")
        ax.plot(x, scipy.stats.norm.pdf(x, mean_bg, std_bg), 'r-', label='Grass'
    if i == 0 else "")
        ax.set_title(f'Feature {i}', fontsize=10)
        ax.tick_params(labelsize=8)

    fig.legend(loc='upper right')
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.savefig(output_path, dpi=150)
    plt.close(fig)


def plot_segmentation_results(predicted_mask, true_mask, title, output_path):
    """Plot segmentation results."""
    fig, axes = plt.subplots(1, 2, figsize=(12, 6))
    axes[0].imshow(predicted_mask, cmap='gray')
    axes[0].set_title('Predicted Mask', fontsize=14)
    axes[0].axis('off')
    axes[1].imshow(true_mask, cmap='gray')
    axes[1].set_title('Ground Truth Mask', fontsize=14)
    axes[1].axis('off')
    plt.suptitle(title, fontsize=16)
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.savefig(output_path, dpi=150)
    plt.close(fig)


def extract_dct_vectors_sliding_window(image, zig_zag_map):
    """
    Processes an image using a sliding window.
    Returns a (H*W, 64) array of DCT vectors.
    """
    img = np.float32(image)
```

```python
    h, w = img.shape

    img_padded = np.pad(img, ((0, BLOCK_SIZE - 1), (0, BLOCK_SIZE - 1)), mode='reflect')
    all_dct_vectors = np.zeros((h * w, BLOCK_SIZE * BLOCK_SIZE))

    idx = 0
    for i in tqdm(range(h), desc="Extracting DCT"):
        for j in range(w):
            block = img_padded[i:i + BLOCK_SIZE, j:j + BLOCK_SIZE]
            block_dct = dct2(block)
            all_dct_vectors[idx] = block_dct.flatten()[zig_zag_map]
            idx += 1

    return all_dct_vectors, h, w


def classify_blocks_vectorized(X_test, mean_fg, cov_fg, mean_bg, cov_bg, log_prior_fg, log_prior_bg):
    """
    Classifies a set of test vectors using Bayesian decision rule.
    """
    log_ll_fg = scipy.stats.multivariate_normal.logpdf(X_test, mean=mean_fg, cov=cov_fg, allow_singular=True)
    log_ll_bg = scipy.stats.multivariate_normal.logpdf(X_test, mean=mean_bg, cov=cov_bg, allow_singular=True)

    g_fg = log_ll_fg + log_prior_fg
    g_bg = log_ll_bg + log_prior_grass
    decisions = (g_fg > g_bg).astype(int)
    return decisions


def compute_error_rate(predicted_mask, true_mask, prior_fg, prior_bg):
    """
    Compute Bayesian probability of error (weighted).
    P(error) = P(error|cheetah)P(cheetah) + P(error|grass)P(grass)
    """
    predicted_binary = (predicted_mask > 0.5).astype(int)
    true_binary = (true_mask > 0.5).astype(int)

    fg_pixels_total = np.sum(true_binary == 1)
    bg_pixels_total = np.sum(true_binary == 0)

    p_error_given_fg = 0.0
    if fg_pixels_total > 0:
        fg_misclassified = np.sum((true_binary == 1) & (predicted_binary == 0))
        p_error_given_fg = fg_misclassified / fg_pixels_total

    p_error_given_bg = 0.0
    if bg_pixels_total > 0:
        bg_misclassified = np.sum((true_binary == 0) & (predicted_binary == 1))
        p_error_given_bg = bg_misclassified / bg_pixels_total

    total_error = (p_error_given_fg * prior_fg) + (p_error_given_bg * prior_bg)
    return total_error


def main():
```

```
    os.makedirs(OUTPUT_DIR, exist_ok=True)

    zig_zag_map = load_zig_zag_map(ZIG_ZAG_FILE)
    train_data = scipy.io.loadmat(DATA_FILE)
    fg_data = train_data['TrainsampleDCT_FG']
    bg_data = train_data['TrainsampleDCT_BG']

    image = imageio.imread(IMAGE_FILE, mode='L')
    true_mask = imageio.imread(MASK_FILE)
    true_mask = (true_mask > 127).astype(int)

    n_fg = fg_data.shape[0]
    n_bg = bg_data.shape[0]
    n_total = n_fg + n_bg
    prior_cheetah = n_fg / n_total
    prior_grass = n_bg / n_total
    log_prior_cheetah = np.log(prior_cheetah)
    log_prior_grass = np.log(prior_grass)

    print("\n" + "=" * 70)
    print("Problem 6(a): Prior Probabilities")
    print(f"P(Y=cheetah): {prior_cheetah:.6f} (N={n_fg})")
    print(f"P(Y=grass):   {prior_grass:.6f} (N={n_bg})")

    print("\n" + "=" * 70)
    print("Problem 6(b): ML Parameters and Feature Selection")

    mean_fg_64, cov_fg_64 = compute_mle_parameters(fg_data)
    mean_bg_64, cov_bg_64 = compute_mle_parameters(bg_data)

    best_8_indices, worst_8_indices = select_features_kl(fg_data, bg_data)

    print(f"Best 8 features (KL Div): {best_8_indices.tolist()}")
    print(f"Worst 8 features (KL Div): {worst_8_indices.tolist()}")

    plot_features_filename = os.path.join(OUTPUT_DIR, 'best_worst_8_features_KL.png')
    plot_best_worst_features(fg_data, bg_data, best_8_indices, worst_8_indices,
                             plot_features_filename)
    print(f"Feature plots saved to {plot_features_filename}")

    plot_all_filename = os.path.join(OUTPUT_DIR, 'marginal_densities_all_64.png')
    plot_all_features(fg_data, bg_data, plot_all_filename)
    print(f"All 64 feature plots saved to {plot_all_filename}")


    print("\n" + "=" * 70)
    print("Problem 6(c): Classification")
    X_test_64, img_h, img_w = extract_dct_vectors_sliding_window(image, zig_zag_map)

    print("Classifying with 64D Gaussians...")
    decisions_64_flat = classify_blocks_vectorized(X_test_64,
                                                   mean_fg_64, cov_fg_64,
                                                   mean_bg_64, cov_bg_64,
                                                   log_prior_cheetah, log_prior_grass)
    mask_64d = decisions_64_flat.reshape(img_h, img_w)
```

```python
    error_64d = compute_error_rate(mask_64d, true_mask, prior_cheetah, prior_grass)

    plot_64d_filename = os.path.join(OUTPUT_DIR, 'segmentation_64d_KL.png')
    plot_segmentation_results(mask_64d, true_mask,
                              f'64D Gaussian (KL) (Error: {error_64d:.4f})',
                              plot_64d_filename)
    print(f"64D plot saved to {plot_64d_filename}")

    print("Classifying with 8D Gaussians (best features)...")
    X_test_8 = X_test_64[:, best_8_indices]

    mean_fg_8 = mean_fg_64[best_8_indices]
    mean_bg_8 = mean_bg_64[best_8_indices]
    cov_fg_8 = cov_fg_64[np.ix_(best_8_indices, best_8_indices)]
    cov_bg_8 = cov_bg_64[np.ix_(best_8_indices, best_8_indices)]

    decisions_8_flat = classify_blocks_vectorized(X_test_8,
                                                  mean_fg_8, cov_fg_8,
                                                  mean_bg_8, cov_bg_8,
                                                  log_prior_cheetah, log_prior_grass)
    mask_8d = decisions_8_flat.reshape(img_h, img_w)
    error_8d = compute_error_rate(mask_8d, true_mask, prior_cheetah, prior_grass)

    plot_8d_filename = os.path.join(OUTPUT_DIR, 'segmentation_8d_KL.png')
    plot_segmentation_results(mask_8d, true_mask,
                              f'8D Gaussian (KL) (Error: {error_8d:.4f})',
                              plot_8d_filename)
    print(f"8D plot saved to {plot_8d_filename}")

    print("\n" + "=" * 70)
    print("Final Explanation of Results")
    print("=" * 70)
    print(f"64D Classifier Bayesian Error: {error_64d:.6f}")
    print(f" 8D Classifier Bayesian Error: {error_8d:.6f}")

    if error_8d < error_64d:
        print("\nSUCCESS!")
    else:
        print("\nSomething went wrong!")

if __name__ == "__main__":
    main()
```

Listing 1: Python code for HW2 (hw2/hw2_solution.py)