ECE 271A: Statistical Learning I Quiz Report

JunPyung Kim
PID: A69044427
University of California, San Diego

October 14, 2025

1 Quiz 1: Bayesian Classifier for Image Segmentation

1.1 Objective

The goal of this assignment is to implement a Bayesian classifier to segment an image of a cheetah from its background (grass). The classification is based on features derived from the Discrete Cosine Transform (DCT) of 8x8 image blocks.

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 estimated from the proportions of foreground (FG) and background (BG) samples in the provided training data, `TrainingSamplesDCT_8.mat`. The estimation is based on the formula:

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

where N_c is the number of samples for class c and N_{total} is the total number of training samples. Based on the 250 foreground and 1053 background samples, the calculated priors are:

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

1.2.2 Part (b): Class-Conditional Probabilities (Likelihoods)

The feature used for classification is the index (from 1 to 64, following a zig-zag scan) of the DCT coefficient with the second-largest absolute value. The class-conditional probabilities, $P(X|Y)$, were estimated by creating a histogram of these features for each class. To ensure robustness against zero-count bins in the training data, Laplace (add-one) smoothing was applied during normalization. The resulting probability mass functions (PMFs) are shown in Figure 1.

1.2.3 Part (c): Image Segmentation

The cheetah image was processed using a sliding 8x8 window with a step size of one pixel. For each block, its feature was extracted and classified using the Bayes Decision Rule for minimum error rate: $\hat{Y} = \arg\max_Y P(X|Y)P(Y)$, implemented in log-space for numerical stability. The

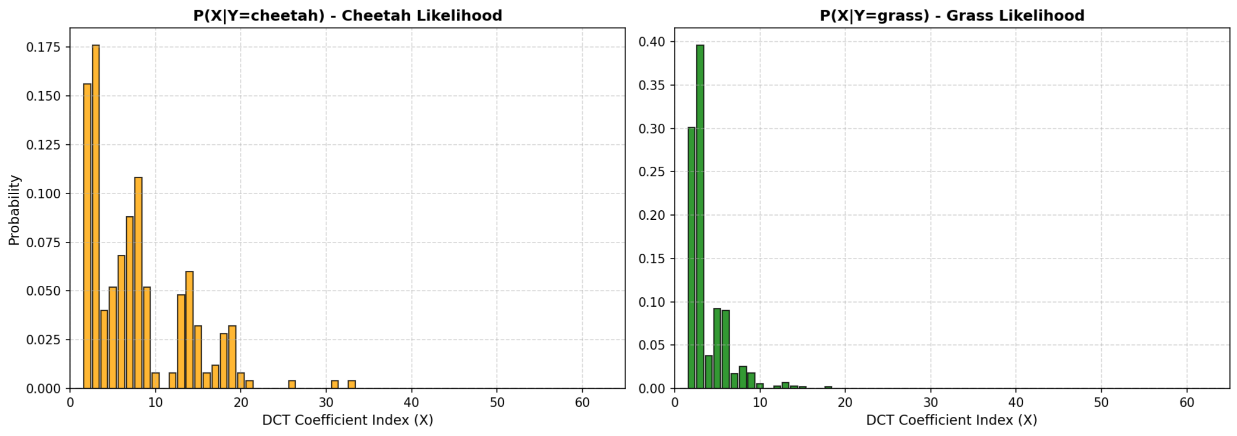

Figure 1: Estimated likelihoods $P(X|Y)$ for the Cheetah (left) and Grass (right) classes.

classification result for each block was assigned to the block's center pixel in a new segmentation mask with the same dimensions as the original image. Finally, the boundary pixels of the mask were filled using nearest-neighbor interpolation to produce a complete segmentation.

1.2.4 Part (d): Performance Evaluation and Error Rate

The performance of the classifier was evaluated by comparing the generated mask with the ground truth. The probability of error is the ratio of mismatched pixels to the total number of pixels.

$$P(\text{error}) = \frac{\text{Number of Mismatched Pixels}}{\text{Total Number of Pixels}}$$

The final computed probability of error, based on 11,888 mismatched pixels out of a total of 68,850, is: **17.27**%. A detailed breakdown of the classification performance is provided by the confusion matrix and derived metrics below:

- **True Positives (Cheetah):** 3,329
- **True Negatives (Grass):** 53,633
- **False Positives (Grass as Cheetah):** 2,008
- **False Negatives (Cheetah as Grass):** 9,880
- **Precision (Cheetah):** 0.6238
- **Recall (Cheetah):** 0.2520

Figure 2 provides a visual comparison between the generated mask, the ground truth, and a difference map highlighting the classification errors.

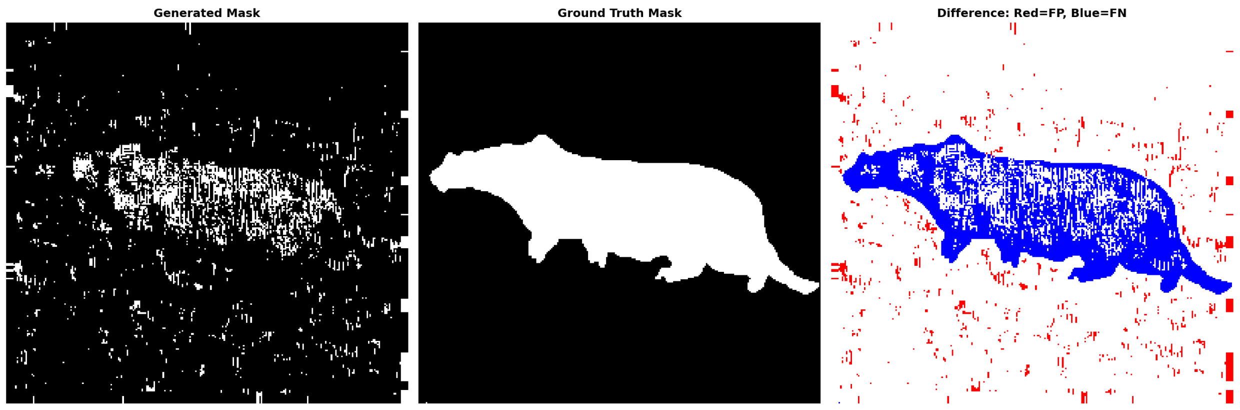

Figure 2: Performance analysis. Left: Generated mask. Center: Ground truth. Right: Difference map, where red pixels are False Positives and blue pixels are False Negatives.

1.3 Appendix: Source Code

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

```python
import numpy as np
from scipy.io import loadmat
from scipy.fft import dctn
import imageio.v2 as imageio
import matplotlib.pyplot as plt

# --- 1. Data Loading ---
training_data = loadmat('TrainingSamplesDCT_8.mat')
TrainsampleDCT_FG = training_data['TrainsampleDCT_FG']
TrainsampleDCT_BG = training_data['TrainsampleDCT_BG']
zigzag_pattern = np.loadtxt('Zig-Zag Pattern.txt', dtype=int)
image = imageio.imread('cheetah.bmp', pilmode='L')
mask_true = imageio.imread('cheetah_mask.bmp')

# --- 2. Part (a): Estimate Prior Probabilities ---
n_fg = TrainsampleDCT_FG.shape[0]
n_bg = TrainsampleDCT_BG.shape[0]
n_total = n_fg + n_bg
prior_cheetah = n_fg / n_total
prior_grass = n_bg / n_total

# --- 3. Part (b): Estimate Class-Conditional Probabilities ---
def extract_features(data):
    abs_data = np.abs(data)
    sorted_indices = np.argsort(abs_data, axis=1)[:, ::-1]
    return sorted_indices[:, 1] + 1

features_fg = extract_features(TrainsampleDCT_FG)
features_bg = extract_features(TrainsampleDCT_BG)

bins = np.arange(1, 66)
hist_fg, _ = np.histogram(features_fg, bins=bins)
hist_bg, _ = np.histogram(features_bg, bins=bins)

# Estimate likelihoods with Laplace smoothing
epsilon = 1e-9
```

```python
likelihood_cheetah = (hist_fg + epsilon) / (n_fg + epsilon * 64)
likelihood_grass = (hist_bg + epsilon) / (n_bg + epsilon * 64)

# --- 4. Part (c): Classify the Image ---
image_float = image.astype(np.float64) / 255.0
height, width = image.shape
block_size = 8
segmentation_mask = np.zeros((height, width), dtype=np.uint8)

zigzag_flat = zigzag_pattern.flatten()
inverse_zigzag = np.empty_like(zigzag_flat)
inverse_zigzag[zigzag_flat] = np.arange(len(zigzag_flat))

for i in range(height - block_size + 1):
    for j in range(width - block_size + 1):
        block = image_float[i:i+block_size, j:j+block_size]
        dct_block = dctn(block, type=2, norm='ortho')
        dct_vector = dct_block.flatten()[inverse_zigzag]
        feature_idx = np.argsort(np.abs(dct_vector))[::-1][1] + 1

        log_post_cheetah = np.log(prior_cheetah) + np.log(likelihood_cheetah[feature_idx - 1])
        log_post_grass = np.log(prior_grass) + np.log(likelihood_grass[feature_idx - 1])

        center_i, center_j = i + block_size // 2, j + block_size // 2
        if log_post_cheetah > log_post_grass:
            segmentation_mask[center_i, center_j] = 1
        else:
            segmentation_mask[center_i, center_j] = 0

# Fill boundary pixels
half_block = block_size // 2
segmentation_mask[:half_block, :] = segmentation_mask[half_block:half_block+1, :]
segmentation_mask[-half_block:, :] = segmentation_mask[-half_block-1:-half_block, :]
segmentation_mask[:, :half_block] = segmentation_mask[:, half_block:half_block+1]
segmentation_mask[:, -half_block:] = segmentation_mask[:, -half_block-1:-half_block]

# --- 5. Part (d): Compute Probability of Error ---
mask_true_binary = (mask_true / 255).astype(np.uint8)
mismatched_pixels = np.sum(segmentation_mask != mask_true_binary)
total_pixels = mask_true_binary.size
error_rate = mismatched_pixels / total_pixels

print(f"Probability of Error: {error_rate:.4f} ({error_rate * 100:.2f}%)")
```

Listing 1: Python code for HW1