ECE 271A: Statistical Learning I Quiz Report

JunPyung Kim
PID: A69044427
University of California, San Diego

November 21, 2025

1 Quiz 3: Bayesian Parameter Estimation

1.1 Objective

The objective of this assignment is to explore Bayesian Parameter Estimation by treating the parameters of the class-conditional densities as random variables. We compare three different estimators:

1. **Maximum Likelihood (ML) Solution:** Computes parameters solely from the training data.
2. **Maximum A Posteriori (MAP) Solution:** Estimates the parameter vector that maximizes the posterior density.
3. **Bayesian Predictive Solution:** Marginals out the unknown parameters to compute the predictive distribution $P(x|\mathcal{D})$.

We analyze the performance of these estimators as a function of the prior uncertainty (α) and the training set size (N), using two different strategies for the prior mean.

1.2 Methodology

1.2.1 Model Setup

We model the class-conditional densities as multivariate Gaussians. For this assignment, we assume the covariance Σ is known (approximated by the sample covariance of the training set), but the mean μ is unknown. The prior distribution for the mean is modeled as a Gaussian:

$$P(\mu) = \mathcal{G}(\mu, \mu_0, \Sigma_0)$$

where $\Sigma_0 = \alpha\mathbf{W}$. The parameter α scales the variance of the prior; a small α indicates high confidence in the prior mean μ_0, while a large α indicates high uncertainty.

1.2.2 Strategies

Two strategies were used to select the prior mean μ_0:

- **Strategy 1 (Informative Prior):** μ_0 is set closer to the true distribution (smaller for Cheetah, larger for Grass). This represents a "good" guess.
- **Strategy 2 (Non-Informative/Poor Prior):** μ_0 is set to the middle of the dynamic range for both classes. This represents a "bad" or generic guess.

1.3 Results

The Probability of Error (PoE) was computed for varying values of α across four datasets ($\mathcal{D}_1$ to $\mathcal{D}_4$) of increasing size.

1.3.1 Strategy 1: Informative Prior

In Strategy 1, the prior information is accurate. As observed in Figures 1 and 2, the Bayesian and MAP estimators achieve a lower probability of error than the ML estimator when α is small (high confidence in the good prior). As α increases, the estimators converge to the ML solution.

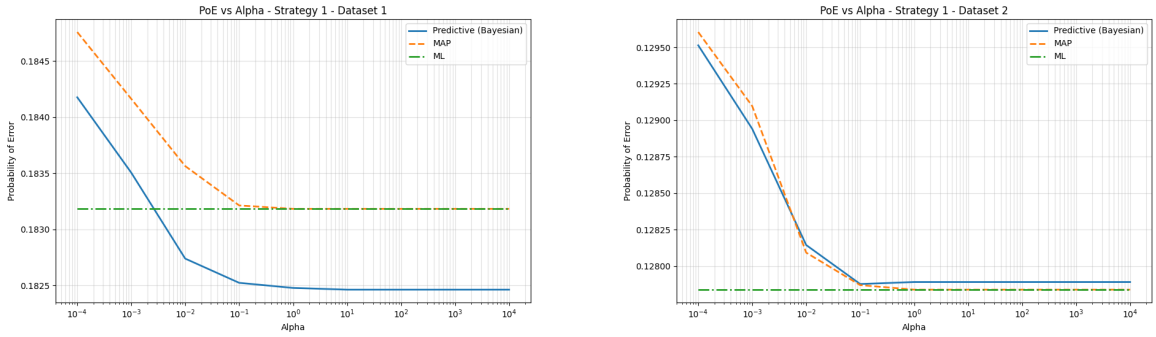

Figure 1: Strategy 1 Results for Dataset 1 (Left) and Dataset 2 (Right).

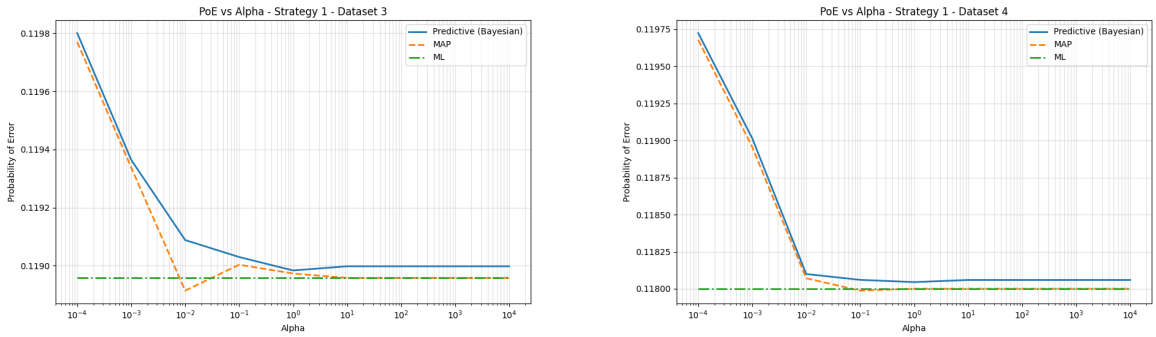

Figure 2: Strategy 1 Results for Dataset 3 (Left) and Dataset 4 (Right).

1.3.2 Strategy 2: Non-Informative Prior

In Strategy 2, the prior information is poor (it does not distinguish between classes). As shown in Figures 3 and 4, the Bayesian and MAP estimators perform worse than the ML estimator when α is small. The prior pulls the decision boundary away from the optimal location. However, as α increases, the data dominates the prior, and the performance converges to the ML baseline.

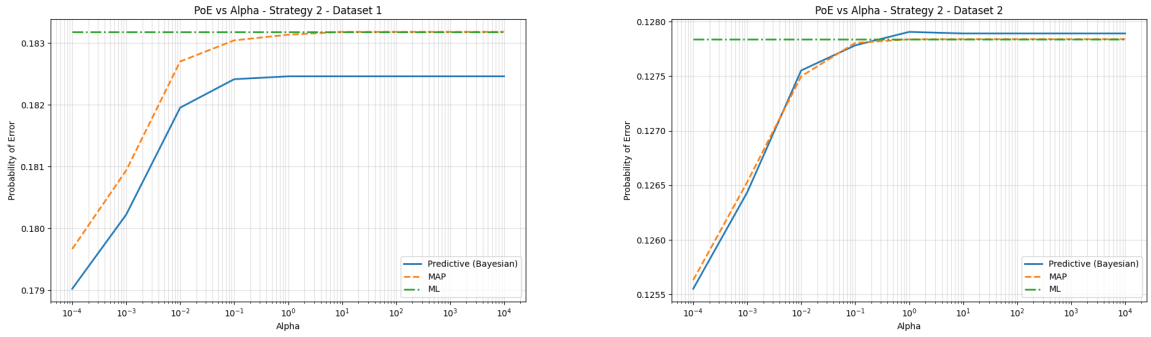

Figure 3: Strategy 2 Results for Dataset 1 (Left) and Dataset 2 (Right).

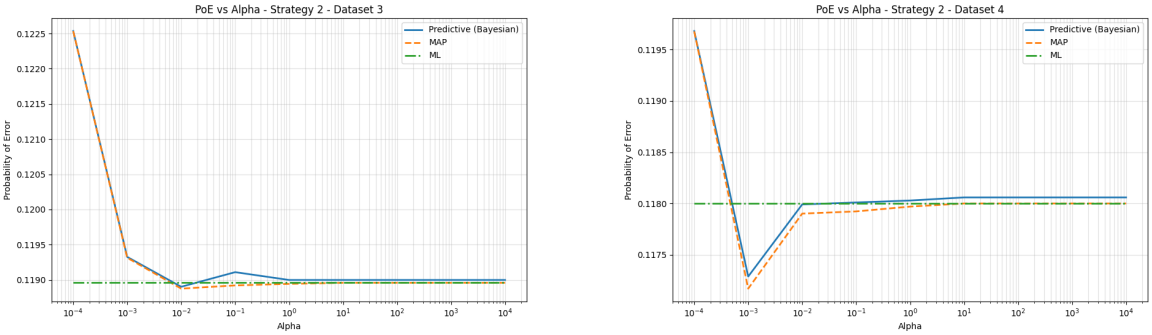

Figure 4: Strategy 2 Results for Dataset 3 (Left) and Dataset 4 (Right).

1.4 Discussion

1.4.1 Relative Behavior of Estimators

The ML solution (Green line) appears as a flat line because it does not depend on the prior parameter α. The MAP (Orange dashed) and Bayesian Predictive (Blue) solutions vary with α.

- When $\alpha \to 0$, the prior dominates. If the prior is good (Strategy 1), PoE is low. If the prior is bad (Strategy 2), PoE is high.
- When $\alpha \to \infty$, the prior variance becomes infinite (flat prior), and the Bayesian/MAP estimates converge to the ML estimate.

The Bayesian Predictive classifier generally closely tracks the MAP estimate in performance, though theoretically, it accounts for the uncertainty in μ by integrating it out rather than just picking the peak.

1.4.2 Effect of Dataset Size

Comparing $\mathcal{D}_1$ (Smallest) to $\mathcal{D}_4$ (Largest):

- For smaller datasets ($\mathcal{D}_1$), the prior has a stronger and longer-lasting influence. The convergence to the ML line happens more slowly as α increases.
- For larger datasets ($\mathcal{D}_4$), the likelihood term (data) dominates the posterior density much faster. The curves converge to the ML solution at smaller values of α compared to $\mathcal{D}_1$. This illustrates that more data reduces the sensitivity to the choice of prior.

1.5 Appendix: Source Code

The Python source code used to generate these results is included below.

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

BLOCK_SIZE = 8
DATA_DIR = 'data/'
OUTPUT_DIR = 'hw3/output/'
CHEETAH_IMG = os.path.join(DATA_DIR, 'cheetah.bmp')
CHEETAH_MASK = os.path.join(DATA_DIR, 'cheetah_mask.bmp')
ZIG_ZAG_FILE = os.path.join(DATA_DIR, 'Zig-Zag Pattern.txt')

os.makedirs(OUTPUT_DIR, exist_ok=True)
warnings.filterwarnings('ignore')

def load_mat_file(filename):
    return scipy.io.loadmat(os.path.join(DATA_DIR, filename))

def load_zig_zag_map(filepath):
    zig_zag_pattern = np.loadtxt(filepath, dtype=int)
    return np.argsort(zig_zag_pattern.flatten())

def dct2(block):
    return dctn(block, type=2, norm='ortho')

def extract_dct_features(image_path, zig_zag_map):
    """
    Reads image, applies sliding window, computes DCT, returns (N, 64) feature
    matrix.
    """
    img = imageio.imread(image_path, mode='L')
    img = np.float32(img) / 255.0
    h, w = img.shape

    img_padded = np.pad(img, ((0, BLOCK_SIZE-1), (0, BLOCK_SIZE-1)), mode='
    constant')

    num_vectors = h * w
    features = np.zeros((num_vectors, 64))

    idx = 0
    for r in range(h):
        for c in range(w):
```

```
            block = img_padded[r:r+BLOCK_SIZE, c:c+BLOCK_SIZE]
            dct_block = dct2(block).flatten()
            features[idx] = dct_block[zig_zag_map]
            idx += 1

    return features, h, w

def compute_error(predictions, ground_truth, prior_fg, prior_bg):
    """
    Computes probability of error.
    """
    gt_fg = (ground_truth == 255)
    gt_bg = (ground_truth == 0)

    error_fg = np.sum(predictions[gt_fg] == 0) / np.sum(gt_fg)
    error_bg = np.sum(predictions[gt_bg] == 1) / np.sum(gt_bg)

    total_error = error_fg * prior_fg + error_bg * prior_bg
    return total_error

def gaussian_log_likelihood_batch(X, mu, cov):
    """
    Computes log N(x; mu, cov) for a batch of X (N, 64).
    """
    cov = cov + np.eye(64) * 1e-10
    sign, logdet = np.linalg.slogdet(cov)
    if sign <= 0:
        logdet = np.log(np.linalg.det(cov) + 1e-20)

    inv_cov = np.linalg.inv(cov)
    diff = X - mu
    mahal = np.sum((diff @ inv_cov) * diff, axis=1)

    const = -0.5 * (64 * np.log(2 * np.pi) + logdet)
    return const - 0.5 * mahal

def get_posterior_params(mu_0, cov_0, data, N):
    """
    Calculates mu_n and Sigma_n based on Bayesian update formulas.
    """
    mu_ml = np.mean(data, axis=0)
    sigma_ml = np.cov(data, rowvar=False, ddof=0)

    inv_term = np.linalg.inv(cov_0 + (1/N) * sigma_ml)
    sigma_n = cov_0 @ inv_term @ ((1/N) * sigma_ml)

    term1 = cov_0 @ inv_term @ mu_ml
    term2 = ((1/N) * sigma_ml) @ inv_term @ mu_0
    mu_n = term1 + term2

    return mu_n, sigma_n, mu_ml, sigma_ml

def solve():
    print("Loading Data...")
    zig_zag = load_zig_zag_map(ZIG_ZAG_FILE)
    alpha_mat = load_mat_file('Alpha.mat')['alpha'].flatten()
    train_subsets = load_mat_file('TrainingSamplesDCT_subsets_8.mat')
    prior_1 = load_mat_file('Prior_1.mat')
    prior_2 = load_mat_file('Prior_2.mat')
```

```
    print("Processing Test Image...")
    test_features, h_img, w_img = extract_dct_features(CHEETAH_IMG, zig_zag)
    ground_truth = imageio.imread(CHEETAH_MASK).flatten()

    strategies = [
        {'name': 'Strategy 1', 'data': prior_1},
        {'name': 'Strategy 2', 'data': prior_2}
    ]

    datasets = [
        {'id': 1, 'bg': train_subsets['D1_BG'], 'fg': train_subsets['D1_FG']},
        {'id': 2, 'bg': train_subsets['D2_BG'], 'fg': train_subsets['D2_FG']},
        {'id': 3, 'bg': train_subsets['D3_BG'], 'fg': train_subsets['D3_FG']},
        {'id': 4, 'bg': train_subsets['D4_BG'], 'fg': train_subsets['D4_FG']},
    ]

    print("Starting Classification Loop...")

    for strat in strategies:
        strat_name = strat['name']
        strat_data = strat['data']
        W0 = strat_data['W0'].flatten()
        mu0_FG = strat_data['mu0_FG'].flatten()
        mu0_BG = strat_data['mu0_BG'].flatten()

        print(f"\n--- Processing {strat_name} ---")

        for dataset in datasets:
            d_id = dataset['id']
            fg_train = dataset['fg']
            bg_train = dataset['bg']

            n_fg = fg_train.shape[0]
            n_bg = bg_train.shape[0]

            p_fg = n_fg / (n_fg + n_bg)
            p_bg = n_bg / (n_fg + n_bg)
            log_p_fg = np.log(p_fg)
            log_p_bg = np.log(p_bg)

            errors_ml = []
            errors_map = []
            errors_bayes = []

            # 1. ML Solution
            mu_ml_fg = np.mean(fg_train, axis=0)
            cov_ml_fg = np.cov(fg_train, rowvar=False, ddof=0)
            mu_ml_bg = np.mean(bg_train, axis=0)
            cov_ml_bg = np.cov(bg_train, rowvar=False, ddof=0)

            ll_fg_ml = gaussian_log_likelihood_batch(test_features, mu_ml_fg,
    cov_ml_fg)
            ll_bg_ml = gaussian_log_likelihood_batch(test_features, mu_ml_bg,
    cov_ml_bg)
            decisions_ml = ((ll_fg_ml + log_p_fg) > (ll_bg_ml + log_p_bg)).astype(
    int)
            err_ml_val = compute_error(decisions_ml, ground_truth, p_fg, p_bg)

```

```
            print(f"Dataset {d_id}: Processing alphas...")
            for alpha in tqdm(alpha_mat, leave=False):
                errors_ml.append(err_ml_val)
                cov_0 = np.diag(alpha * W0)

                mu_n_fg, sigma_n_fg, _, _ = get_posterior_params(mu0_FG, cov_0, fg_train, n_fg)
                mu_n_bg, sigma_n_bg, _, _ = get_posterior_params(mu0_BG, cov_0, bg_train, n_bg)

                # 2. MAP Solution
                ll_fg_map = gaussian_log_likelihood_batch(test_features, mu_n_fg, cov_ml_fg)
                ll_bg_map = gaussian_log_likelihood_batch(test_features, mu_n_bg, cov_ml_bg)
                decisions_map = ((ll_fg_map + log_p_fg) > (ll_bg_map + log_p_bg)).astype(int)
                errors_map.append(compute_error(decisions_map, ground_truth, p_fg, p_bg))

                # 3. Bayesian Predictive Solution
                pred_cov_fg = cov_ml_fg + sigma_n_fg
                pred_cov_bg = cov_ml_bg + sigma_n_bg

                ll_fg_bayes = gaussian_log_likelihood_batch(test_features, mu_n_fg, pred_cov_fg)
                ll_bg_bayes = gaussian_log_likelihood_batch(test_features, mu_n_bg, pred_cov_bg)
                decisions_bayes = ((ll_fg_bayes + log_p_fg) > (ll_bg_bayes + log_p_bg)).astype(int)
                errors_bayes.append(compute_error(decisions_bayes, ground_truth, p_fg, p_bg))

            plt.figure(figsize=(10, 6))
            plt.semilogx(alpha_mat, errors_bayes, label='Predictive (Bayesian)', linewidth=2)
            plt.semilogx(alpha_mat, errors_map, label='MAP', linewidth=2, linestyle='--')
            plt.semilogx(alpha_mat, errors_ml, label='ML', linewidth=2, linestyle='-.')

            plt.title(f'PoE vs Alpha - {strat_name} - Dataset {d_id}')
            plt.xlabel('Alpha'); plt.ylabel('Probability of Error'); plt.legend()
            plt.grid(True, which="both", ls="-", alpha=0.4)

            plot_filename = f"PoE_Strategy_{strat_name[-1]}_Dataset_{d_id}.png"
            plt.savefig(os.path.join(OUTPUT_DIR, plot_filename))
            plt.close()

if __name__ == '__main__':
    solve()
```

Listing 1: Python code for HW3