ECE 271A: Statistical Learning I Quiz Report

JunPyung Kim
PID: A69044427
University of California, San Diego

December 17, 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 $\Sigma_0 = \alpha\mathbf{W}$ and $\mathbf{W}$ is diagonal (i.e., $\Sigma_0 = \alpha\,\mathrm{diag}(\mathbf{W}_0)$). 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.

Decision rule and evaluation. Class priors in the decision rule are the ML estimates from the training set. The Probability of Error (PoE) is computed on the *cheetah test image* as the *pixel-wise misclassification rate* using the provided ground-truth mask.

1.2.2 Strategies

Two strategies were used to select the prior mean μ_0 (provided in `Prior_1.mat` and `Prior_2.mat`):

- **Strategy 1 (More informative prior):** μ_0 is closer to the training-data ML means for each class.
- **Strategy 2 (Less informative / poorer prior):** μ_0 is farther from the ML means (more generic).

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. In all figures, the ML curve is constant w.r.t. α, and the MAP/Predictive curves converge to the ML baseline as α becomes large.

1.3.1 Strategy 1: More informative prior

Figures 1 and 2 show the results for Strategy 1. For small α, MAP and Bayesian Predictive solutions are influenced strongly by the prior. Depending on how well μ_0 aligns with the test-image distribution, this can slightly improve or slightly degrade PoE relative to ML. As α increases, the influence of the prior diminishes and both MAP and Predictive approaches approach the ML baseline.

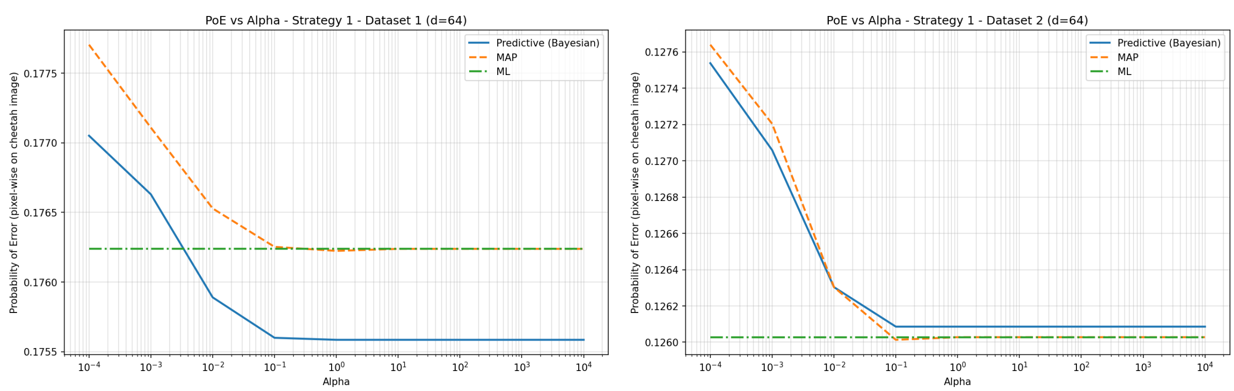

Figure 1: Strategy 1 Results for Dataset 1 (Left) and Dataset 2 (Right), using $d = 64$ DCT coefficients.

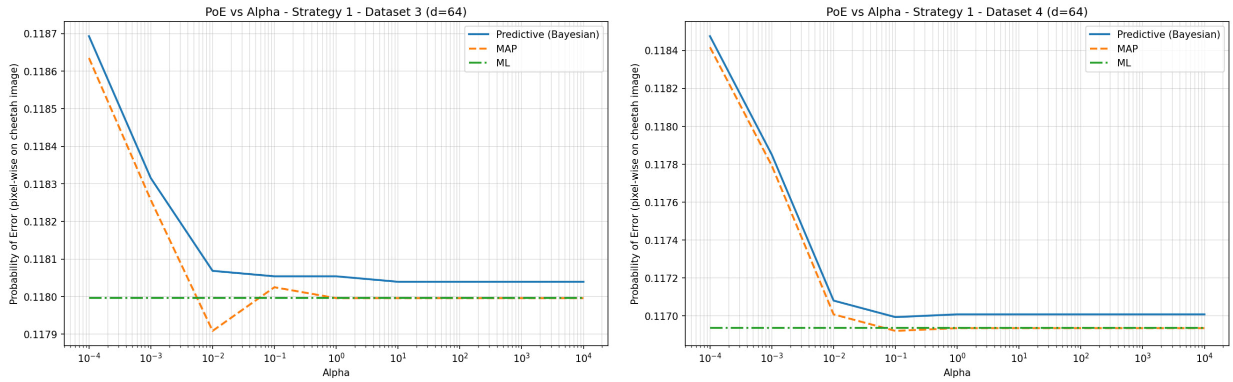

Figure 2: Strategy 1 Results for Dataset 3 (Left) and Dataset 4 (Right), using $d = 64$ DCT coefficients.

1.3.2 Strategy 2: Less informative / poorer prior

Figures 3 and 4 show the results for Strategy 2. With a less accurate prior mean, very small α can pull the posterior mean away from the data-driven estimate and increase PoE. However, for small training sets, a strongly regularizing prior can sometimes reduce variance and yield comparable (or occasionally slightly better) performance than ML at certain α values. As α increases, the curves again converge to the ML baseline.

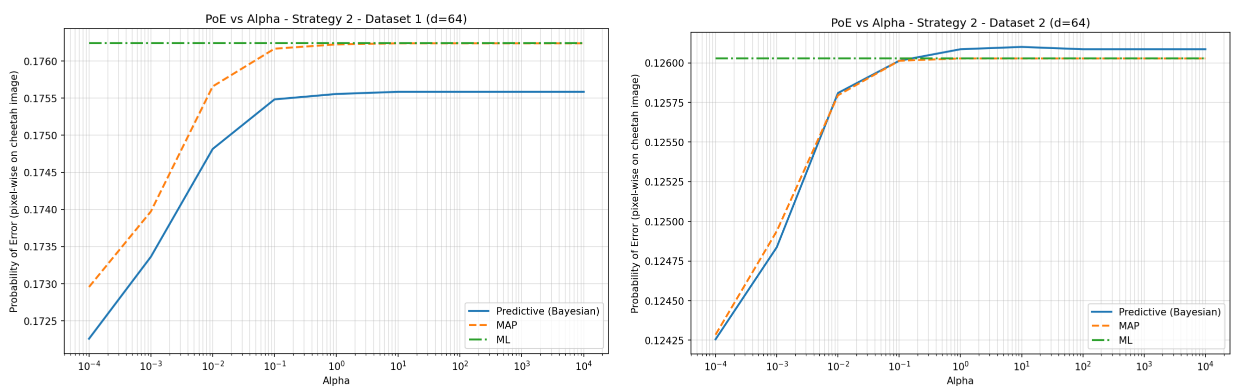

Figure 3: Strategy 2 Results for Dataset 1 (Left) and Dataset 2 (Right), using $d = 64$ DCT coefficients.

1.4 Discussion

1.4.1 Relative Behavior of Estimators

The ML solution appears as a flat line because it does not depend on the prior parameter α. The MAP and Bayesian Predictive solutions vary with α:

- When $\alpha \to 0$, the prior dominates and the posterior mean is pulled toward μ_0. The resulting PoE can be lower or higher than ML depending on how well the prior matches the true/test distribution.

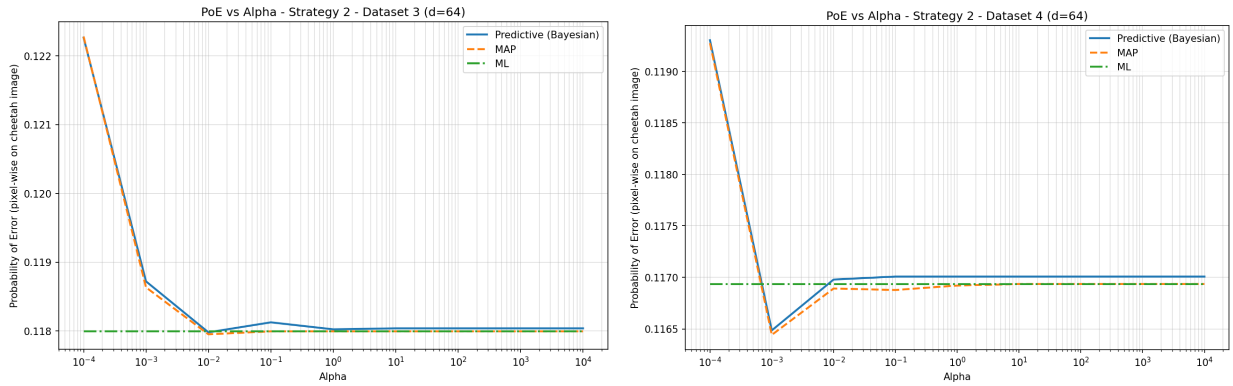

Figure 4: Strategy 2 Results for Dataset 3 (Left) and Dataset 4 (Right), using $d = 64$ DCT coefficients.

- When $\alpha \to \infty$, the prior becomes effectively uninformative, and the MAP estimate of the mean approaches the ML mean (thus the MAP curve approaches the ML baseline).

The Bayesian Predictive classifier typically tracks the MAP curve closely, but can differ slightly because it incorporates posterior uncertainty in μ (through an added covariance term in the predictive distribution).

1.4.2 Effect of Dataset Size

Comparing $\mathcal{D}_1$ (smallest) to $\mathcal{D}_4$ (largest):

- **Higher variance in small datasets:** With fewer training samples, the ML mean estimates are noisier, which generally increases PoE (e.g., the ML baseline is higher on smaller datasets).
- **Prior influence lasts longer for small datasets:** The posterior mean balances μ_0 vs. the sample mean with a term proportional to Σ/N; smaller N makes the data term less concentrated, so the prior can influence the solution over a wider range of α.
- **Faster convergence for larger datasets:** With larger N, the curves converge to ML at smaller α values since the data dominates the posterior more quickly.

1.5 Appendix: Source Code

The Python source code used to generate these results is included below (`hw3/hw3_solution.py`).

```python
#!/usr/bin/env python3
import os
import warnings
from pathlib import Path

import numpy as np
import scipy.io
import imageio.v3 as imageio
import matplotlib
from scipy.fftpack import dctn
from tqdm import tqdm

try:
    matplotlib.use("Agg")
except Exception:
    pass

import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")

BLOCK_SIZE = 8

THIS_DIR = Path(__file__).resolve().parent          # .../hw3
PROJECT_ROOT = THIS_DIR.parent                      # .../ece271a
DATA_DIR = PROJECT_ROOT / "data"
OUTPUT_DIR = THIS_DIR / "output"

CHEETAH_IMG = DATA_DIR / "cheetah.bmp"
CHEETAH_MASK = DATA_DIR / "cheetah_mask.bmp"
ZIG_ZAG_FILE = DATA_DIR / "Zig-Zag Pattern.txt"
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

def load_mat_file(filename: str) -> dict:
    return scipy.io.loadmat(str(DATA_DIR / filename))

def load_zig_zag_map(filepath: Path) -> np.ndarray:
    zig_zag_pattern = np.loadtxt(str(filepath), dtype=int)
    return np.argsort(zig_zag_pattern.flatten())

def dct2(block: np.ndarray) -> np.ndarray:
    return dctn(block, type=2, norm="ortho")

def extract_dct_features(image_path: Path, zig_zag_map: np.ndarray):
    img = imageio.imread(str(image_path), mode="L").astype(np.float32) / 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), dtype=np.float32)

    idx = 0
    for r in range(h):
        for c in range(w):
            block = img_padded[r : r + BLOCK_SIZE, c : c + BLOCK_SIZE]
```

```python
            dct_block = dct2(block).flatten()
            features[idx] = dct_block[zig_zag_map]
            idx += 1
    return features, h, w

def binarize_mask(mask_flat: np.ndarray) -> np.ndarray:
    return (mask_flat >= 128).astype(np.int32)

def compute_poe_pixelwise(pred_labels: np.ndarray, gt_labels01: np.ndarray) ->
    float:
    return float(np.mean(pred_labels != gt_labels01))

def gaussian_log_likelihood_batch(X: np.ndarray, mu: np.ndarray, cov: np.ndarray,
    eps: float = 1e-8) -> np.ndarray:
    d = X.shape[1]
    cov = cov + np.eye(d) * eps
    sign, logdet = np.linalg.slogdet(cov)
    if sign <= 0:
        logdet = np.log(np.linalg.det(cov) + 1e-30)
    inv_cov = np.linalg.inv(cov)

    diff = X - mu
    mahal = np.sum((diff @ inv_cov) * diff, axis=1)
    const = -0.5 * (d * np.log(2 * np.pi) + logdet)
    return const - 0.5 * mahal

def posterior_for_mean(mu0: np.ndarray, Sigma0: np.ndarray, X: np.ndarray):
    N = X.shape[0]
    xbar = np.mean(X, axis=0)
    Sigma = np.cov(X, rowvar=False, ddof=0)

    d = X.shape[1]
    Sigma = Sigma + np.eye(d) * 1e-8
    Sigma_over_N = Sigma / max(N, 1)

    A = Sigma0 + Sigma_over_N
    A_inv = np.linalg.inv(A)

    Sigma_n = Sigma0 @ A_inv @ Sigma_over_N
    mu_n = (Sigma0 @ A_inv @ xbar) + (Sigma_over_N @ A_inv @ mu0)
    return mu_n, Sigma_n

def solve():
    print("[INFO] Loading data...")
    zig_zag = load_zig_zag_map(ZIG_ZAG_FILE)

    alpha_vec = 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("[INFO] Processing test image / mask...")
    test_features_64, _, _ = extract_dct_features(CHEETAH_IMG, zig_zag)
    gt_mask_flat = imageio.imread(str(CHEETAH_MASK)).flatten()
    gt01 = binarize_mask(gt_mask_flat)

    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"]},
    ]

    dims_to_run = [64]

    print("[INFO] Starting classification loop...")
    for strat in strategies:
        strat_name = strat["name"]
        strat_data = strat["data"]

        W0_64 = strat_data["W0"].flatten()
        mu0_FG_64 = strat_data["mu0_FG"].flatten()
        mu0_BG_64 = strat_data["mu0_BG"].flatten()

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

        for d in dims_to_run:
            test_features = test_features_64[:, :d]
            W0 = W0_64[:d]
            mu0_FG = mu0_FG_64[:d]
            mu0_BG = mu0_BG_64[:d]

            for dataset in datasets:
                d_id = dataset["id"]
                fg_train = dataset["fg"][:, :d]
                bg_train = dataset["bg"][:, :d]

                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 + 1e-30)
                log_p_bg = np.log(p_bg + 1e-30)

                mu_ml_fg = np.mean(fg_train, axis=0)
                cov_ml_fg = np.cov(fg_train, rowvar=False, ddof=0) + np.eye(d) * 1e-8
                mu_ml_bg = np.mean(bg_train, axis=0)
                cov_ml_bg = np.cov(bg_train, rowvar=False, ddof=0) + np.eye(d) * 1e-8

                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)
                pred_ml = ((ll_fg_ml + log_p_fg) > (ll_bg_ml + log_p_bg)).astype(np.int32)
                poe_ml = compute_poe_pixelwise(pred_ml, gt01)

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

```

```python
                print(f"[INFO] {strat_name} | Dataset {d_id} | d={d}: sweeping alpha...")
                for alpha in tqdm(alpha_vec, leave=False):
                    Sigma0 = np.diag(alpha * W0 + 1e-30)

                    mu_n_fg, Sigma_n_fg = posterior_for_mean(mu0_FG, Sigma0, fg_train)
                    mu_n_bg, Sigma_n_bg = posterior_for_mean(mu0_BG, Sigma0, bg_train)

                    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)
                    pred_map = ((ll_fg_map + log_p_fg) > (ll_bg_map + log_p_bg)).astype(np.int32)
                    poe_map = compute_poe_pixelwise(pred_map, gt01)

                    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)
                    pred_bayes = ((ll_fg_bayes + log_p_fg) > (ll_bg_bayes + log_p_bg)).astype(np.int32)
                    poe_bayes = compute_poe_pixelwise(pred_bayes, gt01)

                    errors_ml.append(poe_ml)
                    errors_map.append(poe_map)
                    errors_bayes.append(poe_bayes)

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

                plt.title(f"PoE vs Alpha - {strat_name} - Dataset {d_id} (d={d})")
                plt.xlabel("Alpha")
                plt.ylabel("Probability of Error (pixel-wise on cheetah image)")
                plt.legend()
                plt.grid(True, which="both", ls="-", alpha=0.4)

                plot_filename = f"PoE_Strategy_{strat_name[-1]}_Dataset_{d_id}_d{d}.png"
                plt.savefig(str(OUTPUT_DIR / plot_filename), dpi=150, bbox_inches="tight")
                plt.close()

    print(f"\n[DONE] Plots saved to: {OUTPUT_DIR}")

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

Listing 1: Python code for HW3 (final implementation)