

ECE 271A: Statistical Learning I Quiz Report

JunPyung Kim
PID: A69044427
University of California, San Diego

November 23, 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:** Marginalizes 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.

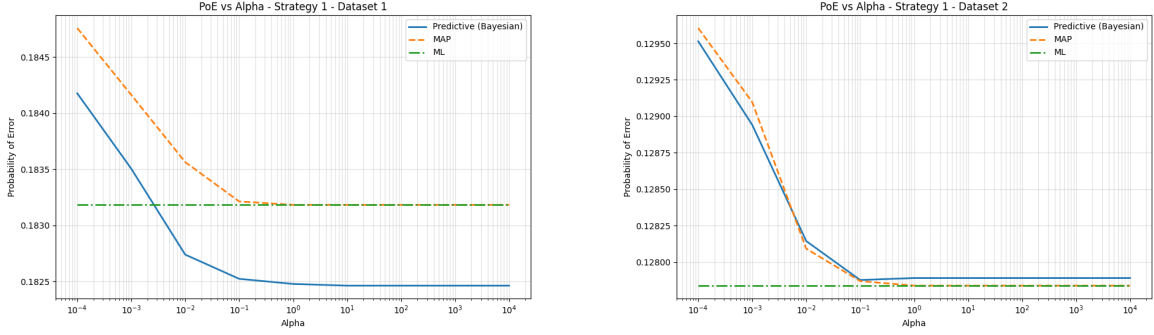


Figure 1: Strategy 1 Results for Dataset 1 (Left) and Dataset 2 (Right).

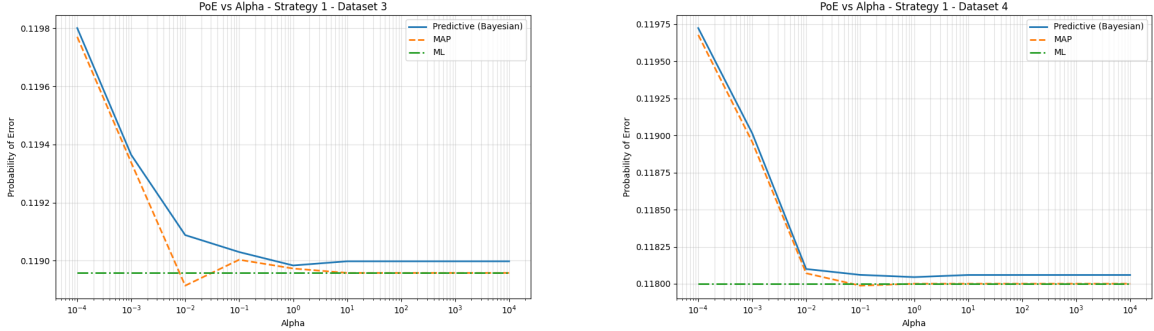


Figure 2: Strategy 1 Results for Dataset 3 (Left) and Dataset 4 (Right).

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.

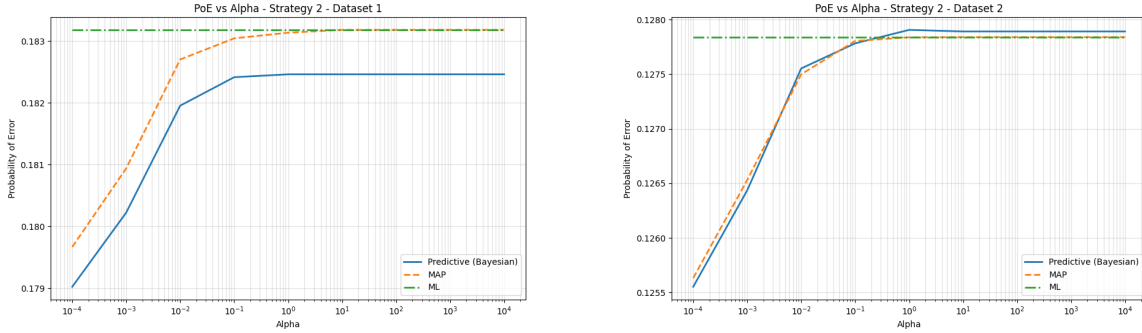


Figure 3: Strategy 2 Results for Dataset 1 (Left) and Dataset 2 (Right).

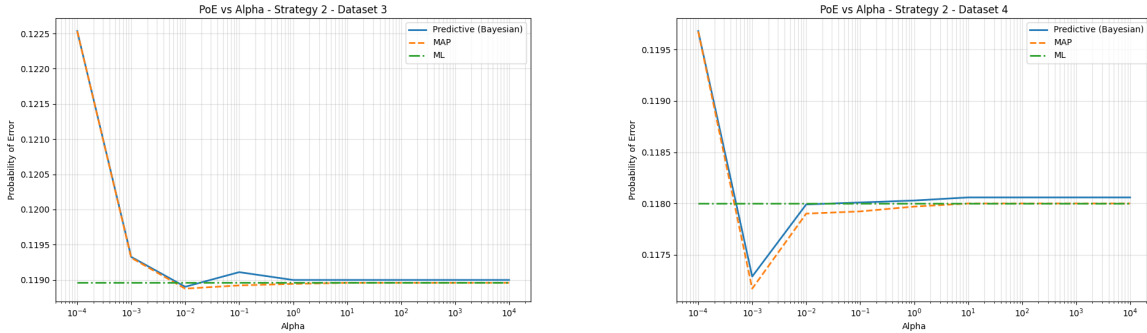


Figure 4: Strategy 2 Results for Dataset 3 (Left) and Dataset 4 (Right).

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 $\alpha \rightarrow 0$, the prior dominates. If the prior is good (Strategy 1), PoE is low. If the prior is bad (Strategy 2), PoE is high.
- When $\alpha \rightarrow \infty$, 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):

- **Overfitting in Small Datasets:** Dataset 1 shows a noticeably higher error rate for the ML solution ($\approx 18.3\%$) compared to other datasets. This is because the number of cheetah samples ($N_{FG} = 75$) is comparable to the feature dimensionality ($d = 64$). This leads to an ill-conditioned covariance estimate and overfitting, which the Bayesian priors help mitigate when α is small.
- **Prior Influence:** 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.
- **Data Dominance:** 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.

```
1 import numpy as np
2 import scipy.io
3 import scipy.stats
4 import imageio.v3 as imageio
5 import matplotlib
6 import os
7 import warnings
8 from scipy.fftpack import dctn
9 from tqdm import tqdm
10
11 try:
12     matplotlib.use('Agg')
13 except ImportError:
14     pass
15
16 import matplotlib.pyplot as plt
17
18 BLOCK_SIZE = 8
19 DATA_DIR = 'data/'
20 OUTPUT_DIR = 'hw3/output/'
21 CHEETAH_IMG = os.path.join(DATA_DIR, 'cheetah.bmp')
22 CHEETAH_MASK = os.path.join(DATA_DIR, 'cheetah_mask.bmp')
23 ZIG_ZAG_FILE = os.path.join(DATA_DIR, 'Zig-Zag Pattern.txt')
24
25 os.makedirs(OUTPUT_DIR, exist_ok=True)
26 warnings.filterwarnings('ignore')
27
28 def load_mat_file(filename):
29     return scipy.io.loadmat(os.path.join(DATA_DIR, filename))
30
31 def load_zig_zag_map(filepath):
32     zig_zag_pattern = np.loadtxt(filepath, dtype=int)
33     return np.argsort(zig_zag_pattern.flatten())
34
35 def dct2(block):
36     return dctn(block, type=2, norm='ortho')
37
38 def extract_dct_features(image_path, zig_zag_map):
39     """
40     Reads image, applies sliding window, computes DCT, returns (N, 64) feature
41     matrix.
42     """
43     img = imageio.imread(image_path, mode='L')
44     img = np.float32(img) / 255.0
45     h, w = img.shape
46
47     img_padded = np.pad(img, ((0, BLOCK_SIZE-1), (0, BLOCK_SIZE-1)), mode='
48     constant')
49
50     num_vectors = h * w
51     features = np.zeros((num_vectors, 64))
52
53     idx = 0
54     for r in range(h):
55         for c in range(w):
```

```

54         block = img_padded[r:r+BLOCK_SIZE, c:c+BLOCK_SIZE]
55         dct_block = dct2(block).flatten()
56         features[idx] = dct_block[zig_zag_map]
57         idx += 1
58
59     return features, h, w
60
61 def compute_error(predictions, ground_truth, prior_fg, prior_bg):
62     """
63     Computes probability of error.
64     """
65     gt_fg = (ground_truth == 255)
66     gt_bg = (ground_truth == 0)
67
68     error_fg = np.sum(predictions[gt_fg] == 0) / np.sum(gt_fg)
69     error_bg = np.sum(predictions[gt_bg] == 1) / np.sum(gt_bg)
70
71     total_error = error_fg * prior_fg + error_bg * prior_bg
72     return total_error
73
74 def gaussian_log_likelihood_batch(X, mu, cov):
75     """
76     Computes log N(x; mu, cov) for a batch of X (N, 64).
77     """
78     cov = cov + np.eye(64) * 1e-10
79     sign, logdet = np.linalg.slogdet(cov)
80     if sign <= 0:
81         logdet = np.log(np.linalg.det(cov) + 1e-20)
82
83     inv_cov = np.linalg.inv(cov)
84     diff = X - mu
85     mahal = np.sum((diff @ inv_cov) * diff, axis=1)
86
87     const = -0.5 * (64 * np.log(2 * np.pi) + logdet)
88     return const - 0.5 * mahal
89
90 def get_posterior_params(mu_0, cov_0, data, N):
91     """
92     Calculates mu_n and Sigma_n based on Bayesian update formulas.
93     """
94     mu_m1 = np.mean(data, axis=0)
95     sigma_m1 = np.cov(data, rowvar=False, ddof=0)
96
97     inv_term = np.linalg.inv(cov_0 + (1/N) * sigma_m1)
98     sigma_n = cov_0 @ inv_term @ ((1/N) * sigma_m1)
99
100     term1 = cov_0 @ inv_term @ mu_m1
101     term2 = ((1/N) * sigma_m1) @ inv_term @ mu_0
102     mu_n = term1 + term2
103
104     return mu_n, sigma_n, mu_m1, sigma_m1
105
106 def solve():
107     print("Loading Data...")
108     zig_zag = load_zig_zag_map(ZIG_ZAG_FILE)
109     alpha_mat = load_mat_file('Alpha.mat')['alpha'].flatten()
110     train_subsets = load_mat_file('TrainingSamplesDCT_subsets_8.mat')
111     prior_1 = load_mat_file('Prior_1.mat')
112     prior_2 = load_mat_file('Prior_2.mat')

```

```

113
114 print("Processing Test Image...")
115 test_features, h_img, w_img = extract_dct_features(CHEETAH_IMG, zig_zag)
116 ground_truth = imageio.imread(CHEETAH_MASK).flatten()
117
118 strategies = [
119     {'name': 'Strategy 1', 'data': prior_1},
120     {'name': 'Strategy 2', 'data': prior_2}
121 ]
122
123 datasets = [
124     {'id': 1, 'bg': train_subsets['D1_BG'], 'fg': train_subsets['D1_FG']},
125     {'id': 2, 'bg': train_subsets['D2_BG'], 'fg': train_subsets['D2_FG']},
126     {'id': 3, 'bg': train_subsets['D3_BG'], 'fg': train_subsets['D3_FG']},
127     {'id': 4, 'bg': train_subsets['D4_BG'], 'fg': train_subsets['D4_FG']},
128 ]
129
130 print("Starting Classification Loop...")
131
132 for strat in strategies:
133     strat_name = strat['name']
134     strat_data = strat['data']
135     W0 = strat_data['W0'].flatten()
136     mu0_FG = strat_data['mu0_FG'].flatten()
137     mu0_BG = strat_data['mu0_BG'].flatten()
138
139     print(f"\n--- Processing {strat_name} ---")
140
141     for dataset in datasets:
142         d_id = dataset['id']
143         fg_train = dataset['fg']
144         bg_train = dataset['bg']
145
146         n_fg = fg_train.shape[0]
147         n_bg = bg_train.shape[0]
148
149         p_fg = n_fg / (n_fg + n_bg)
150         p_bg = n_bg / (n_fg + n_bg)
151         log_p_fg = np.log(p_fg)
152         log_p_bg = np.log(p_bg)
153
154         errors_ml = []
155         errors_map = []
156         errors_bayes = []
157
158         # 1. ML Solution
159         mu_ml_fg = np.mean(fg_train, axis=0)
160         cov_ml_fg = np.cov(fg_train, rowvar=False, ddof=0)
161         mu_ml_bg = np.mean(bg_train, axis=0)
162         cov_ml_bg = np.cov(bg_train, rowvar=False, ddof=0)
163
164         ll_fg_ml = gaussian_log_likelihood_batch(test_features, mu_ml_fg,
165 cov_ml_fg)
166         ll_bg_ml = gaussian_log_likelihood_batch(test_features, mu_ml_bg,
167 cov_ml_bg)
168         decisions_ml = ((ll_fg_ml + log_p_fg) > (ll_bg_ml + log_p_bg)).astype(
169 int)
170         err_ml_val = compute_error(decisions_ml, ground_truth, p_fg, p_bg)

```

```

169         print(f"Dataset {d_id}: Processing alphas...")
170         for alpha in tqdm(alpha_mat, leave=False):
171             errors_ml.append(err_ml_val)
172             cov_0 = np.diag(alpha * W0)
173
174             mu_n_fg, sigma_n_fg, _, _ = get_posterior_params(mu0_FG, cov_0,
175             fg_train, n_fg)
176             mu_n_bg, sigma_n_bg, _, _ = get_posterior_params(mu0_BG, cov_0,
177             bg_train, n_bg)
178
179             # 2. MAP Solution
180             ll_fg_map = gaussian_log_likelihood_batch(test_features, mu_n_fg,
181             cov_ml_fg)
182             ll_bg_map = gaussian_log_likelihood_batch(test_features, mu_n_bg,
183             cov_ml_bg)
184             decisions_map = ((ll_fg_map + log_p_fg) > (ll_bg_map + log_p_bg)).
185             astype(int)
186             errors_map.append(compute_error(decisions_map, ground_truth, p_fg,
187             p_bg))
188
189             # 3. Bayesian Predictive Solution
190             pred_cov_fg = cov_ml_fg + sigma_n_fg
191             pred_cov_bg = cov_ml_bg + sigma_n_bg
192
193             ll_fg_bayes = gaussian_log_likelihood_batch(test_features, mu_n_fg,
194             , pred_cov_fg)
195             ll_bg_bayes = gaussian_log_likelihood_batch(test_features, mu_n_bg,
196             , pred_cov_bg)
197             decisions_bayes = ((ll_fg_bayes + log_p_fg) > (ll_bg_bayes +
198             log_p_bg)).astype(int)
199             errors_bayes.append(compute_error(decisions_bayes, ground_truth,
200             p_fg, p_bg))
201
202             plt.figure(figsize=(10, 6))
203             plt.semilogx(alpha_mat, errors_bayes, label='Predictive (Bayesian)',
204             linewidth=2)
205             plt.semilogx(alpha_mat, errors_map, label='MAP', linewidth=2,
206             linestyle='--')
207             plt.semilogx(alpha_mat, errors_ml, label='ML', linewidth=2, linestyle=
208             '-.')
209
210             plt.title(f'PoE vs Alpha - {strat_name} - Dataset {d_id}')
211             plt.xlabel('Alpha'); plt.ylabel('Probability of Error'); plt.legend()
212             plt.grid(True, which="both", ls="-", alpha=0.4)
213
214             plot_filename = f"PoE_Strategy_{strat_name[-1]}_Dataset_{d_id}.png"
215             plt.savefig(os.path.join(OUTPUT_DIR, plot_filename))
216             plt.close()
217
218 if __name__ == '__main__':
219     solve()

```

Listing 1: Python code for HW3