3F8: Inference
Full Technical Report

Author's Name

February 12, 2026

Abstract

This report implements Bayesian logistic classification with an RBF feature expansion and uses the Laplace approximation to obtain a tractable Gaussian approximation to the weight posterior. The MAP solution is found by optimising the log-posterior, and the Laplace approximation is used for approximate Bayesian prediction and for evidence-based tuning of (σ_0^2, l) via a 10×10 grid search visualised as a heat map. Evidence-based tuning produces cleaner predictive contours and improves predictive log-likelihood relative to the untuned baseline, while hard classification changes only slightly.

1 Introduction

This report studies Bayesian binary classification with logistic regression and makes posterior inference tractable via the Laplace approximation. Under a Gaussian prior, the logistic likelihood is non-conjugate, so the exact posterior is intractable and an approximation is required to obtain uncertainty-aware predictive probabilities.

Following the 3F8 pipeline, I (i) compute the MAP estimate by optimising the log-posterior, (ii) form a Laplace approximation around the MAP point to obtain an approximate posterior covariance and predictive distribution, and (iii) tune the RBF hyper-parameters by maximising a Laplace approximation to the model evidence on a 10×10 grid. The RBF basis enables non-linear decision boundaries, and evidence-based tuning of (σ_0^2, l) (visualised by heat maps) yields a clearer predictive contour plot and improved predictive log-likelihood compared to the baseline hyper-parameter setting.

2 Exercise a)

In this section, we describe how the Laplace approximation is applied to the Bayesian logistic classification model in order to obtain an approximate posterior distribution over the model parameters and an approximate predictive distribution.

Logistic classification model

We consider a binary classification problem with targets $y^{(n)} \in \{0, 1\}$ and feature vectors $\boldsymbol{\phi}^{(n)}$. The likelihood under the logistic regression model is given by

$$p\left(y^{(n)} = 1 \mid \boldsymbol{\phi}^{(n)}, \boldsymbol{\beta}\right) = \sigma\left(\boldsymbol{\beta}^\top \boldsymbol{\phi}^{(n)}\right),$$

where $\sigma(a) = (1 + e^{-a})^{-1}$ is the logistic sigmoid function. Assuming independent observations, the likelihood of the dataset $\mathcal{D} = \{\boldsymbol{\phi}^{(n)}, y^{(n)}\}_{n=1}^N$ is

$$p(\mathbf{y} \mid \boldsymbol{\beta}) = \prod_{n=1}^{N} \sigma\left(\boldsymbol{\beta}^\top \boldsymbol{\phi}^{(n)}\right)^{y^{(n)}} \left(1 - \sigma\left(\boldsymbol{\beta}^\top \boldsymbol{\phi}^{(n)}\right)\right)^{1-y^{(n)}}.$$

Gaussian prior and Laplace approximation

A zero-mean isotropic Gaussian prior is placed over the model parameters,

$$p(\boldsymbol{\beta}) = \mathcal{N}\big(\boldsymbol{\beta} \mid \mathbf{0}, \sigma_0^2 \mathbf{I}\big) \,.$$

The posterior distribution $p(\boldsymbol{\beta} \mid \mathcal{D}) \propto p(\mathbf{y} \mid \boldsymbol{\beta}) p(\boldsymbol{\beta})$ is not analytically tractable due to the non-conjugacy of the logistic likelihood.

The Laplace approximation approximates the posterior by a Gaussian distribution centred at the maximum a posteriori (MAP) estimate $\boldsymbol{\beta}_{\mathrm{MAP}}$,

$$q(\boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\beta} \mid \boldsymbol{\beta}_{\mathrm{MAP}}, \mathbf{S}_N) \,,$$

where $\mathbf{S}_N$ is given by the inverse Hessian of the negative log-posterior evaluated at $\boldsymbol{\beta}_{\mathrm{MAP}}$,

$$\mathbf{S}_N^{-1} = -\nabla^2_{\boldsymbol{\beta}} \log p(\boldsymbol{\beta} \mid \mathcal{D})\big|_{\boldsymbol{\beta}=\boldsymbol{\beta}_{\mathrm{MAP}}} .$$

Approximation of the model evidence

The model evidence (normalization constant in Bayes' rule) is given by

$$p(\mathbf{y}) = \int p(\mathbf{y} \mid \boldsymbol{\beta}) p(\boldsymbol{\beta}) \, d\boldsymbol{\beta}.$$

Under the Laplace approximation, this integral can be approximated by

$$\log p(\mathbf{y}) \approx \log p(\mathbf{y} \mid \boldsymbol{\beta}_{\mathrm{MAP}}) + \log p(\boldsymbol{\beta}_{\mathrm{MAP}}) + \frac{M}{2} \log(2\pi) - \frac{1}{2} \log \left|\mathbf{S}_N^{-1}\right| ,$$

where M is the dimensionality of $\boldsymbol{\beta}$. This quantity is used later for hyper-parameter selection.

Predictive distribution

For a new input $\boldsymbol{\phi}_*$, the predictive distribution is obtained by marginalising over the posterior distribution of the model parameters,

$$p(y_* = 1 \mid \boldsymbol{\phi}_*, \mathcal{D}) = \int \sigma\Big(\boldsymbol{\beta}^\top \boldsymbol{\phi}_*\Big) \, q(\boldsymbol{\beta}) \, d\boldsymbol{\beta}.$$

This integral is intractable. Following Section 4.5.2 of Bishop's book, the logistic sigmoid is approximated by a scaled probit function, which leads to the closed-form approximation

$$p(y_* = 1 \mid \boldsymbol{\phi}_*, \mathcal{D}) \approx \sigma\big(\kappa(\sigma_a^2) \, \mu_a\big) \,,$$

where

$$\mu_a = \boldsymbol{\beta}_{\mathrm{MAP}}^\top \boldsymbol{\phi}_*, \qquad \sigma_a^2 = \boldsymbol{\phi}_*^\top \mathbf{S}_N \boldsymbol{\phi}_*,$$

and the variance-dependent scaling factor is given by

$$\kappa(\sigma_a^2) = \left(1 + \frac{\pi \sigma_a^2}{8}\right)^{-1/2} .$$

3 Exercise b)

Posterior Gradient and Prediction Rules

In Exercise B, the MAP estimate w_{MAP} is obtained by maximising the log-posterior

$$\log p(w \mid \mathcal{D}) \;=\; \log p(y \mid X, w) + \log p(w),$$

with a zero-mean isotropic Gaussian prior $p(w) = \mathcal{N}(0, \alpha_{\text{prior}}^{-1} I)$. Let $a = \tilde{X}w$ and $\sigma(a) = \frac{1}{1+\exp(-a)}$. The gradient of the log-posterior is

$$\nabla_w \log p(w \mid \mathcal{D}) = \tilde{X}^\top\big(y - \sigma(\tilde{X}w)\big) - \alpha_{\text{prior}}\, w.$$

To compute w_{MAP}, `scipy.optimize.fmin_l_bfgs_b` is used, which performs gradient-based *minimisation*. Therefore, I minimise the negative log-posterior, i.e. I pass $-\log p(w \mid \mathcal{D})$ as the objective and $-\nabla_w \log p(w \mid \mathcal{D})$ as its gradient. For completeness, I also implemented an iterative gradient-ascent solver for the same objective; it is interchangeable with L-BFGS-B and serves as a consistency check.

Given w_{MAP}, I compare: (i) the plug-in MAP predictive $p_{\text{MAP}}(y = 1 \mid x) = \sigma(\tilde{x}^\top w_{\text{MAP}})$, and (ii) the Laplace predictive, which accounts for parameter uncertainty via $S_N = H^{-1}$, where $H = \tilde{X}^\top R\tilde{X} + \alpha_{\text{prior}} I$ and $R = \text{diag}(p_n(1 - p_n))$. The Laplace predictive uses the standard logistic–Gaussian approximation $\sigma(\kappa\, \mu)$ with $\kappa(v) = (1 + \frac{\pi}{8}v)^{-1/2}$. Average log-likelihood (train/test) and probability contour plots are used for comparison.

The full runnable implementation (including all calls, plots, and printed outputs) is available at https://github.com/Minghong-Zhao-mz492/cambridge-3f8-inference-bayesian-logistic-laplace.

Python Code (Key Functions)

Posterior Objective, Gradient, Hessian, and Laplace Predictive

```python
from scipy.optimize import fmin_l_bfgs_b

def compute_grad_log_posterior(w, X_tilde, y, alpha_prior):
    p = predict(X_tilde, w)
    return X_tilde.T @ (y - p) - alpha_prior * w

def hessian_neg_log_posterior(w_map, X_tilde, alpha_prior):
    a = X_tilde @ w_map
    p = logistic(a)
    r = p * (1 - p)
    XR = X_tilde * r[:, None]
    H = X_tilde.T @ XR + alpha_prior * np.eye(X_tilde.shape[1])
    return H

def kappa_from_var(var_a):
    return 1.0 / np.sqrt(1.0 + (np.pi / 8.0) * var_a)

def predict_laplace(X_tilde_star, w_map, S_N):
    mu_a = X_tilde_star @ w_map
    var_a = np.sum((X_tilde_star @ S_N) * X_tilde_star, axis=1)
    kappa = kappa_from_var(var_a)
    return logistic(kappa * mu_a)

def compute_average_ll_laplace(X_tilde, y, w, S_N):
    output_prob = predict_laplace(X_tilde, w, S_N)
    eps = 1e-12
    output_prob = np.clip(output_prob, eps, 1 - eps)
    return np.mean(y * np.log(output_prob) + (1 - y) * np.log(1.0 - output_prob))

def plot_predictive_distribution_laplace(X, y, w_map, S_N, map_inputs=lambda x: x):
    xx, yy = plot_data_internal(X, y)
    ax = plt.gca()
    X_grid = np.concatenate((xx.ravel().reshape((-1, 1)), yy.ravel().reshape((-1, 1))), 1)
    X_tilde_grid = get_x_tilde(map_inputs(X_grid))
    Z = predict_laplace(X_tilde_grid, w_map, S_N).reshape(xx.shape)
    cs2 = ax.contour(xx, yy, Z, cmap=’RdBu’, linewidths=2)
    plt.clabel(cs2, fmt=’%2.1f’, colors=’k’, fontsize=14)
    plt.show()

def predict_map(X_tilde_star, w_map):
    a = X_tilde_star @ w_map
    return logistic(a)  #  logistic()

def plot_predictive_distribution_map(X, y, w_map, map_inputs=lambda x: x):
    xx, yy = plot_data_internal(X, y)
    ax = plt.gca()
    X_grid = np.concatenate(
        (xx.ravel().reshape((-1, 1)), yy.ravel().reshape((-1, 1))),
        axis=1
    )
    X_tilde_grid = get_x_tilde(map_inputs(X_grid))
    Z = predict_map(X_tilde_grid, w_map).reshape(xx.shape)

    cs = ax.contour(xx, yy, Z, cmap=’RdBu’, linewidths=2)
    plt.clabel(cs, fmt=’%2.1f’, colors=’k’, fontsize=14)
    plt.show()
```

MAP Solvers (L-BFGS-B; optional gradient ascent)

```
def fit_w_map_lbfgs(X_tilde_train, y_train, X_tilde_test, y_test,
                    alpha_prior, maxiter=200, w0=None):
    D = X_tilde_train.shape[1]
    if w0 is None:
        w0 = np.zeros(D)

    ll_train_hist = []
    ll_test_hist = []

    def f_obj(w):
        return -compute_log_posterior(w, X_tilde_train, y_train, alpha_prior)

    def f_grad(w):
        return -compute_grad_log_posterior(w, X_tilde_train, y_train, alpha_prior)

    def callback(wk):
        ll_train_hist.append(compute_average_ll(X_tilde_train, y_train, wk))
        ll_test_hist.append(compute_average_ll(X_tilde_test, y_test, wk))

    w_map, f_min, info = fmin_l_bfgs_b(
        func=f_obj,
        x0=w0,
        fprime=f_grad,
        maxiter=maxiter,
        callback=callback
    )

    return w_map, np.array(ll_train_hist), np.array(ll_test_hist), info

def fit_w_map_gradient_ascent(X_tilde_train, y_train, X_tilde_test, y_test, n_steps, alpha_lr, alpha_pri
    w = np.zeros(X_tilde_train.shape[1])
    ll_train = np.zeros(n_steps)
    ll_test = np.zeros(n_steps)

    for i in range(n_steps):
        grad = compute_grad_log_posterior(w, X_tilde_train, y_train, alpha_prior)
        w = w + alpha_lr * grad

        ll_train[i] = compute_average_ll(X_tilde_train, y_train, w)
        ll_test[i]  = compute_average_ll(X_tilde_test,  y_test,  w)

    return w, ll_train, ll_test
```

4 Exercise c)

Using the same setting as in the coursework, I expanded the 2D inputs with Gaussian RBF features and fixed the hyperparameters to $\sigma_0^2 = 1$ and $l = 0.1$. I then visualised probability contours for (i) the MAP plug-in predictive distribution (with w_{MAP} obtained via `scipy.optimize.fmin_l_bfgs_b`), and (ii) the full Bayesian predictive distribution under the Laplace approximation around w_{MAP} (Figure 1).

From the plots alone, the two predictions look almost identical: the $p(y = 1 \mid x) = 0.5$ contour (i.e. the decision boundary) changes very little between the MAP and Laplace methods, so the overall classification boundary is hard to distinguish by eye. The more noticeable differences appear in the more extreme probability contours (e.g. around 0.2 and 0.8): compared to the MAP plug-in predictive, the Laplace predictive is slightly less confident, with these contours being marginally closer to 0.5. This is most evident near boundary regions and for more isolated points (outliers), where integrating parameter uncertainty tends to soften the predicted probabilities.

Although these visual differences are subtle, the separation between the two approaches is more clearly reflected in the numerical metrics (average log-likelihood), which are discussed further in Exercise D.

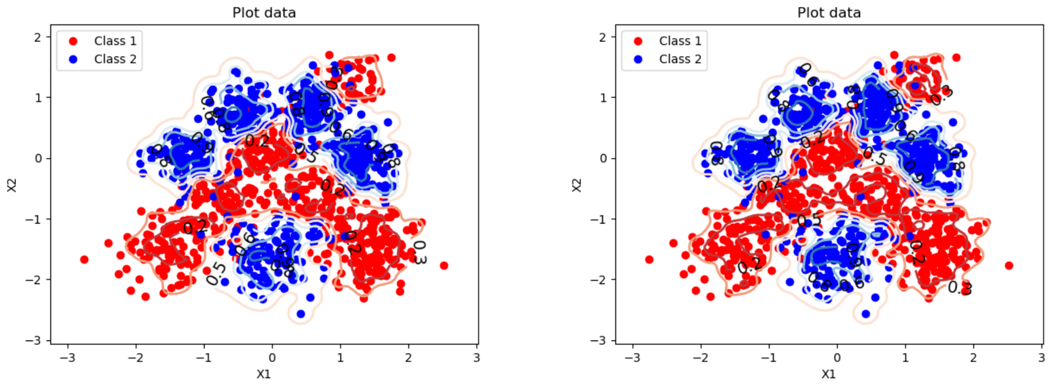

Figure 1: Plots showing data and contour lines for the predictive distribution generated by the Laplace approximation (left) and the MAP solution (right).

5 Exercise d)

Avg. Train ll	**Avg. Test ll**
-0.209463	-0.344511

Table 1: Log-likelihoods for MAP solution.

Avg. Train ll	**Avg. Test ll**
-0.249748	-0.360125

Table 2: Log-likelihoods for Laplace approximation.

		$\hat{y}$	
		0	1
y	0	0.430	0.050
	1	0.045	0.475

Table 3: Conf. matrix for for MAP solution.

		$\hat{y}$	
		0	1
y	0	0.430	0.050
	1	0.045	0.475

Table 4: Conf. matrix for Laplace approximation.

The numerical results are summarised in Tables 1–4. For both training and test sets, the plug-in MAP predictive achieves a higher (less negative) average log-likelihood than the Laplace predictive: MAP $= -0.209$ (train) and -0.345 (test) versus Laplace $= -0.250$ (train) and -0.360 (test). In contrast, the confusion matrices for MAP and Laplace are *exactly identical* (to three decimal places, and in fact unchanged when printing more decimals), meaning that the two methods produce the same hard classifications under the threshold $\hat{y} = \mathbb{I}[p(y = 1 \mid x) > 0.5]$. This matches the visual observation in Exercise C: the $p(y = 1 \mid x) = 0.5$ contour (decision boundary) is essentially unchanged.

The difference therefore lies in the *calibrated probabilities* rather than the predicted labels. Although both approaches classify the same points as 0/1 at the 0.5 threshold, the Laplace predictive integrates parameter uncertainty and slightly shrinks probabilities towards 0.5, leading to different (typically less extreme) probabilities $p(y = 1 \mid x)$. This changes the log-likelihood even when the final classifications are identical. In this run (with $l = 0.1$ and 800/200 train/test split), the Laplace correction reduces the average log-likelihood on both train and test, indicating that the softened probabilities are not beneficial for this particular split/setting, despite leaving the decision boundary unchanged.

6 Exercise e)

In this exercise, I tune the RBF hyper-parameters σ_0^2 (output scale) and l (length-scale) by maximising the Laplace approximation of the model evidence. I use a 10×10 grid search: for each candidate pair (σ_0^2, l), I (i) rebuild the RBF-expanded design matrix $\tilde{X}$ (with centres fixed to the training inputs), (ii) compute the corresponding MAP weights w_{MAP} using `scipy.optimize.fmin_l_bfgs_b` by minimising the negative log-posterior, and then (iii) evaluate the Laplace evidence score at that MAP solution. Repeating this for all 100 grid points yields a matrix of approximate log-evidence values, visualised as a heat map in Figure 2. Brighter regions indicate larger evidence, and hence more preferred hyper-parameters under the Bayesian model selection criterion.

The grid points are chosen on a logarithmic scale (using `logspace`) rather than linear spacing, since σ_0^2 and l are scale parameters and the evidence can vary most significantly across orders of magnitude. A log-spaced grid forms a geometric progression, so adjacent points differ by a constant multiplicative factor; this provides a balanced exploration of both small and large values. In the first (coarse) grid, I use 10 log-spaced values for σ_0^2 between 0.1 and 10, and 10 log-spaced values for l between 0.01 and 0.5. The maximum evidence in this coarse search is attained at

$$\sigma_0^2 \approx 0.774264, \qquad l = 0.5.$$

Although a single 10×10 grid search is sufficient to satisfy the exercise requirements, I additionally performed a second (finer) grid search to further localise the optimum. This refinement step is not required by the coursework instructions, but it provides a more precise estimate of the maximiser of the evidence within the high-evidence region identified by the first heat map. The refined search yields

$$\sigma_0^2 \approx 0.834768, \qquad l \approx 0.528195.$$

In the next exercise, I compare the predictive performance using these tuned hyper-parameters. Empirically, the refinement produces only marginal changes: the confusion matrix remains unchanged (to the reported precision), while the Laplace predictive average log-likelihood changes slightly (train decreases from -0.175738 to -0.175781, and test increases from -0.256043 to -0.255756). This suggests that the predictive decision boundary (and the resulting hard classifications at threshold 0.5) is already stable after the first grid search, and that the second-pass refinement mainly affects predicted probabilities near the boundary rather than the final class assignments.

Core Code for Evidence Grid and Heatmap Computation

The following code block contains the key implementation steps for Exercise (e): evaluating the Laplace-approximated log-evidence over a 10×10 hyper-parameter grid (σ_0^2, l) and identifying the best grid point. For full runnable scripts (including plotting, figure saving, and end-to-end reproduction of all outputs), please refer to my GitHub repository: github.com/Minghong-Zhao-mz492/cambridge-3f8-inference-bayesian-logistic-laplace.

```
def log_evidence_laplace_unnorm(X_tilde_train, y_train, w_map, alpha_prior):
    """
    Laplace log-evidence (marginal likelihood) approximation, up to constants
    independent of (sigma0^2, l) when alpha_prior is fixed:

      log p(D)  log p(y|w_map) + log p(w_map) + (M/2)log(2) - 0.5 log|H|

    where H is the Hessian of the NEGATIVE log-posterior at w_map.
    """
    y_train = np.asarray(y_train).ravel()

    # Log-likelihood at w_map (SUM, not average)
    p = predict(X_tilde_train, w_map)
    eps = 1e-12
    p = np.clip(p, eps, 1 - eps)
    log_lik = np.sum(y_train * np.log(p) + (1 - y_train) * np.log(1 - p))

    # Log-prior at w_map (Gaussian prior, up to additive constant)
    log_prior = -0.5 * alpha_prior * np.dot(w_map, w_map)

    # Hessian of negative log-posterior + stable logdet
    H = hessian_neg_log_posterior(w_map, X_tilde_train, alpha_prior)
    sign, logdetH = np.linalg.slogdet(H)
    if sign <= 0:
        # Numerical guard: add small jitter to encourage positive definiteness
        jitter = 1e-6
        Hj = H + jitter * np.eye(H.shape[0])
        sign, logdetH = np.linalg.slogdet(Hj)
        if sign <= 0:
            return -np.inf

    M = X_tilde_train.shape[1]
    return log_lik + log_prior + 0.5 * M * np.log(2.0 * np.pi) - 0.5 * logdetH

# Choose 10 log-spaced points for the hyperparameters
sigma0_sq_grid = np.logspace(-1, 1, 10)             # 0.1 ... 10
l_grid         = np.logspace(-2, np.log10(0.5), 10) # 0.01 ... 0.5

# Grid search

best_logZ = -np.inf
best_sigma0_sq = None
best_l = None
best_w_map = None

for i, sigma0_sq in enumerate(sigma0_sq_grid):
    for j, l_val in enumerate(l_grid):

        # Phi = sigma0^2 * exp(-||x-z||^2 / (2 l^2)), centres = X_train
        Phi_train = sigma0_sq * evaluate_basis_functions(l_val, X_train, X_train)
        Phi_test  = sigma0_sq * evaluate_basis_functions(l_val, X_test,  X_train)

        X_tilde_train_g = get_x_tilde(Phi_train)
        X_tilde_test_g  = get_x_tilde(Phi_test)

        # MAP fit (reusing your existing solver)
        w_map_g, _, _, info_g = fit_w_map_lbfgs(
            X_tilde_train_g, y_train,
            X_tilde_test_g,  y_test,
            alpha_prior=alpha_prior,
            maxiter=200
        )

        # Laplace log-evidence (approx.)
        logZ = log_evidence_laplace_unnorm(X_tilde_train_g, y_train, w_map_g, alpha_prior)
        evidence_grid[i, j] = logZ

        if logZ > best_logZ:
            best_logZ = logZ
            best_sigma0_sq = sigma0_sq
            best_l = l_val
            best_w_map = w_map_g

        print(f"[grid] sigma0_sq={sigma0_sq:.3g}, l={l_val:.3g} -> logZ{logZ:.6f}")

print("\n[best] sigma0_sq =", best_sigma0_sq)
print("[best] l          =", best_l)
print("[best] logZ       =", best_logZ)

# Heatmap for Figure in Exercise (e)
plot_heatmap(evidence_grid, sigma0_sq_grid, l_grid, save_path="heatmap_evidence.png")
```

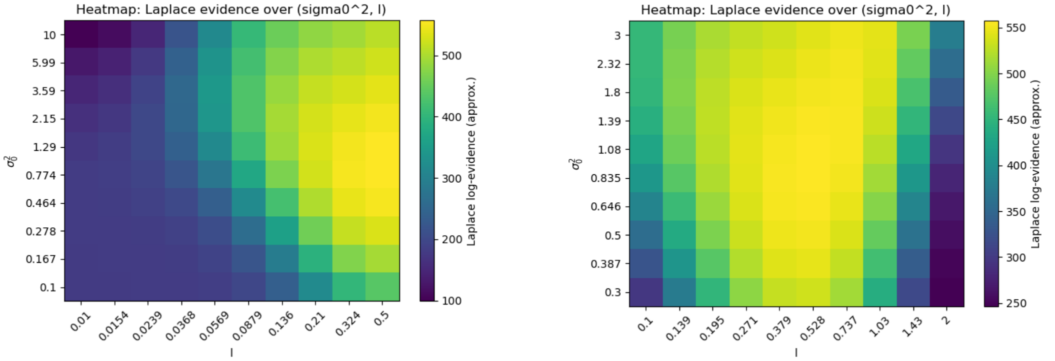

Figure 2: Heat map plots of the (Laplace-approximated) log model evidence over a 10 × 10 grid of hyper-parameters (σ_0^2, l). The left panel uses a coarse grid spanning $\sigma_0^2 \in [0.1, 10]$ and $l \in [0.01, 0.5]$ to obtain an initial overview of the evidence landscape. The right panel refines the search to a narrower region, $\sigma_0^2 \in [0.3, 3]$ and $l \in [0.1, 2]$, yielding a clearer localisation of the high-evidence area used for selecting tuned hyper-parameters.

7 Exercise f)

Figure 3 visualises the class predictive probabilities under the Laplace approximation after tuning the RBF hyper-parameters (σ_0^2, l) by maximising the (Laplace-approximated) model evidence. Relative to the baseline setting in Exercise d, the tuned model produces a noticeably different contour geometry: the $p(y = 1 \mid x) = 0.5$ contour (decision boundary) shifts visibly and forms a more plausible separating curve for the two clusters. In particular, the 0.5 contour now encloses the main red region and includes the red outliers, while the neighbouring confidence contours (e.g. 0.6/0.8 versus 0.2/0.4) appear more consistently aligned with the dense parts of each class. Overall, the tuned contours provide a cleaner and more interpretable predictive surface.

The quantitative performance is summarised in Tables 5 and 6. Evidence-based tuning substantially improves the probabilistic fit compared to Exercise d: the Laplace average log-likelihood increases from -0.249748 to -0.175781 on the training set and from -0.360125 to -0.255756 on the test set (Table 5). This improvement indicates that the tuned model assigns higher probability to the observed labels on average, i.e. the predictive probabilities are better calibrated under the tuned feature geometry.

In contrast, the confusion matrix (Table 6) changes only slightly. Using the 0.5 threshold, the tuned model achieves an accuracy of $0.425 + 0.485 = 0.910$ (91.0%), compared with $0.430 + 0.475 = 0.905$ (90.5%) in Exercise d. The small change in accuracy despite a visibly shifted decision boundary can be explained by the distribution of test points: many points lie with a comfortable margin from the boundary, so moderate boundary movement does not flip their predicted labels. The main effect of tuning is therefore not only on the hard classifications, but on the probability values across the input space (especially away from the boundary), which is precisely what the log-likelihood metric captures.

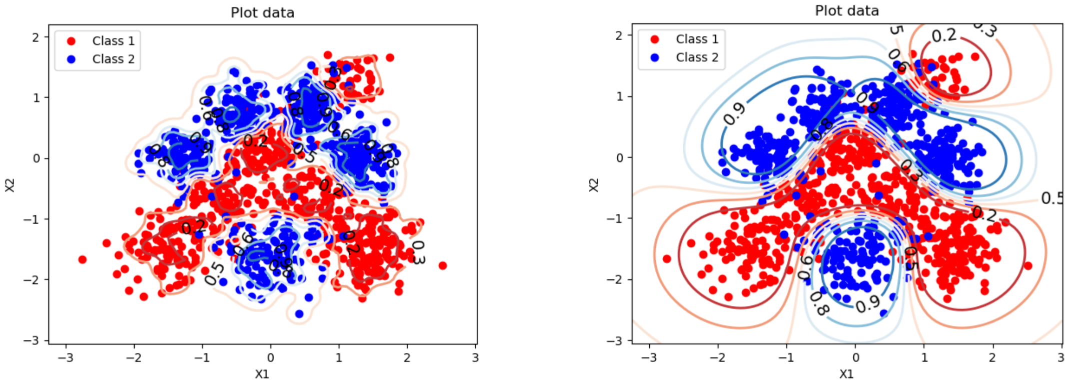

Figure 3: Contours of the Laplace predictive probabilities before hyper-parameter tuning (left, baseline $\sigma_0^2 = 1$, $l = 0.1$) and after evidence-based tuning (right, tuned σ_0^2 and l).

Avg. Train ll	**Avg. Test ll**
-0.175781	-0.255756

Table 5: Average training and test log-likelihoods for Laplace approximation after hyper-parameter tuning by maximising the model evidence.

		$\hat{y}$	
		0	1
y	0	0.425	0.055
	1	0.035	0.485

Table 6: Confusion matrix for Laplace approximation after hyper-parameter tuning by maximising the model evidence.

8 Conclusions

This report implemented Bayesian logistic classification with an RBF basis expansion and compared the plug-in MAP predictor with the Laplace approximation to the posterior predictive distribution. The RBF features are crucial for introducing nonlinearity: by mapping the original 2D inputs into a higher-dimensional space, the classifier can represent a curved decision boundary that separates the two classes much more effectively than a purely linear model.

For a fixed set of hyper-parameters, the MAP weights were obtained by maximising the log-posterior (implemented via L-BFGS-B minimisation of the negative log-posterior). The Laplace approximation then provided a tractable Gaussian approximation to the posterior around the MAP point, enabling an approximate Bayesian predictive distribution that is typically smoother and less over-confident than the plug-in MAP predictive. In practice, the most visible improvement came from selecting appropriate RBF hyper-parameters: tuning (σ_0^2, l) via the Laplace-approximated model evidence and visualising the evidence landscape with a heat map produced a much clearer and more plausible predictive contour plot, and improved the predictive log-likelihood compared to the untuned baseline.

There are several limitations. First, the evidence optimisation used a coarse grid search, so the result is resolution-limited and the optimum may lie between grid points (indeed, a second refined grid only slightly shifted the estimated maximiser). Second, the best l value was close to the boundary of the searched range, which suggests that a wider search interval (or a different optimisation strategy) could be informative. Finally, the Laplace approximation assumes the posterior is approximately Gaussian near the mode; for strongly non-Gaussian posteriors or multi-modal settings this approximation may be inaccurate, although it worked well for this dataset and model.