ENGINEERING TRIPOS PART II A

3F8 **Inference**

Bayesian Logistic Classification

Full Technical Report

Basil Mustafa
Queens' College, Cambridge
(bm490@cam.ac.uk)

1 Introduction

Maximum likelihood logistic classification was implemented as part of the short lab, the report for which is included in Appendix A.

The two most significant drawbacks of maximum likelihood are the overfitting issues faced, particularly when training datasets are small, and the non-uniqueness of solutions. This can be tackled by the inclusion of a **prior distribution** on the model weights which reflects initial assumptions about the weights. This gives the **Maximum A-Posteriori** solution, which is maximum likelihood accounting for these initial assumptions.

Improving on that is the fully Bayesian approach which calculates a predictive distribution for new data by integrating over all possible solutions, weighted by the posterior distribution. In the case of linear Gaussian models, the maximum probability of the Bayesian predictive distribution is the same as the MAP prediction; so this technique is not used to its full extent here in that the final predictions are the same for those two, but it is particularly useful if there are subsequent calculations made from the predictions at which point the information contained in the predictive distribution can prove very useful.

2 Theory

2.1 Model analysis

The model is the same logistic regression model discussed in Appendix A, using radial basis functions parametrised by their variance, l.

Prior Distribution For N ($= |\mathcal{D}_{\text{train}}| + 1$) weights, an uncorrelated Gaussian prior with zero mean and variance σ_0 was used:

$$p(\mathbf{w}) = \frac{1}{\sqrt{(2\pi\sigma_0^2)^N}}\exp(-\frac{1}{2\sigma_0^2}\mathbf{w}^\intercal\mathbf{w})$$

Likelihood The likelihood $p(\mathbf{y}|\mathbf{X}, \mathbf{w})$ is as given in equation 1 of Appendix A

Posterior Distribution By Bayes rule, the posterior on $\mathbf{y}$ is proportional to the product of the prior and the likelihood:

$$p(\mathbf{w}|\mathcal{D}) \propto p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w}) \tag{1}$$

Where $\mathcal{D}$ denotes the training data $\{\mathbf{X}, \mathbf{y}\}$. Taking $\mathcal{L}^*(\mathbf{w})$ to be the terms of the logarithm of (1) which are functions of $\mathbf{w}$ (and thus of interest for minimisation), where $\mathcal{L}(\mathbf{w})$ is the log-likelihood in Equation 2 of Appendix A:

$$\begin{aligned}\mathcal{L}^*(\mathbf{w}) &= \mathcal{L}(\mathbf{w}) - \frac{1}{2\sigma_0^2}\mathbf{w}^\intercal\mathbf{w} \\ &= \sum_{n=1}^{N} y^{(n)}\log(\sigma(\mathbf{w}^\intercal\mathbf{x}^{(n)})) + (1 - y^{(n)})\log(1 - \sigma(\mathbf{w}^\intercal\mathbf{x}^{(n)})) - \frac{1}{2\sigma_0^2}\mathbf{w}^\intercal\mathbf{w}\end{aligned} \tag{2}$$

The gradient of this is:

$$\frac{\partial\mathcal{L}^*(\mathbf{w})}{\partial\mathbf{w}} = \sum_{n=1}^{N}(y^{(n)} - \sigma(\mathbf{w}^\intercal\mathbf{x}^{(n)}))\mathbf{x}^{(n)} - \frac{1}{\sigma_0^2}\mathbf{w} \tag{3}$$

Equations 2 and 3 are used by the optimisation package in order to compute $\mathbf{w}_{\text{MAP}}$.

2.1.1 Laplace Approximation

The fully Bayesian treatment of the problem requires a predictive distribution for a new $\mathbf{y}^\star$ given data $\mathbf{x}^\star$. This requires a typically intractable integral over the posterior, so the posterior $p(\mathbf{w}|\mathbf{x},\mathbf{y})$ is approximated by a Gaussian $q(\mathbf{w}|\mathbf{x},\mathbf{y})$. This is the Laplace approximation.

Developing the approximation Let $p(\mathbf{x})$ be a multivariate distribution which takes the form $p(\mathbf{x}) = \frac{1}{X}f(\mathbf{x})$ where X is the normalisation constant. The goal of the Laplace approximation is to fit a Gaussian to $p(\mathbf{x})$ centred on its *mode*, $\mathbf{x}_0$. Once $\mathbf{x}_0$ is found, the Gaussian's mean is known as this is the centre of its distribution. The covariance $\mathbf{S}_N$ is all that is required to finish characterising the approximation. It is found by considering a truncated Taylor expansion of $\ln(f(\mathbf{x}))$ about the mode:

$$\ln(f(\mathbf{x})) \approx \ln(f(\mathbf{x}_0)) + \mathbf{x}_0^\intercal \nabla f(\mathbf{x}) - \frac{1}{2}(\mathbf{x}-\mathbf{x}_0)^\intercal \mathbf{A}(\mathbf{x_0})(\mathbf{x}-\mathbf{x}_0) + O(\mathbf{x}^3) \tag{4}$$

$\mathbf{A}$ is the negative Hessian matrix, defined as:

$$\mathbf{A} = -\nabla\nabla \ln(f(\mathbf{x}))|_{(x)=(x)_0}$$

By definition of $\mathbf{x_0}$ as the mode of $f(\mathbf{x})$, the second term in Equation 4 is zero. Taking the exponential of Equation 4 to get $f(\mathbf{x})$ back:

$$f(\mathbf{x}) \propto f(\mathbf{x}_0)\exp\Big(-\frac{1}{2}(\mathbf{x}-\mathbf{x}_0)^\intercal \mathbf{A}(\mathbf{x}-\mathbf{x}_0)\Big)$$

This takes the form of a Gaussian. The normalising constant $X = (2\pi)^{N/2} f(\mathbf{x}_0)|\mathbf{A}^{-1}|$. The Laplace approximation of $p(\mathbf{x})$ thus has covariance matrix equal to the inverse of the Hessian:

$$q(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{x}_0, \mathbf{A}^{-1})$$

Applying the Laplace Approximation to MAP Logistic Regression The mode of the $p(\mathbf{w}|\mathcal{D})$ is, by definition, the Maximum A-Posteriori estimate of $\mathbf{w}$ given the data $\mathcal{D}$. The Hessian for the posterior is derived below:

$$\begin{aligned}
\mathbf{A} &= -\nabla(\nabla\mathcal{L}^*(\mathbf{w})^\intercal) \\
&= \frac{1}{\sigma_0^2}\mathbf{I} - \sum_{n=1}^{N} -\nabla\sigma(\mathbf{w}^\intercal\mathbf{x}^{(n)})\mathbf{x}^{(n)\intercal} \\
&= \frac{1}{\sigma_0^2}\mathbf{I} + \sum_{n=1}^{N} \sigma(\mathbf{w}^\intercal\mathbf{x}^{(n)})(1-\sigma(\mathbf{w}^\intercal\mathbf{x}^{(n)}))\mathbf{x}^{(n)}\mathbf{x}^{(n)\intercal}
\end{aligned} \tag{5}$$

Therefore, $p(\mathbf{w}|\mathcal{D}) \approx \mathcal{N}(\mathbf{w}|\mathbf{w}_{\text{MAP}}, \mathbf{A}^{-1})$. The Hessian calculation is shown in Listings 1.

```
def hessian(prior_var, x, w):
    # Prior's contribution - uncorrelated weights/ diagonal cov
    hess = np.eye(x.shape[1]) / prior_var

    # Component due to data
    sig = logistic(x @ w)
    for row, yn in zip(x, sig*(1-sig)):
        hess += yn * np.outer(row, row)

    return hess
```

Listing 1: Function for calculation of the Hessian matrix, $\mathbf{A}$.

2.1.2 Model evidence

The normalisation constant corresponds to the $\frac{1}{p(\mathbf{X})}$ term in Bayes rule missing from Equation 1. Therefore, $p(\mathbf{X}) = X$. More usefully (for computational purposes), the model evidence is given by $\log(p(\mathbf{X})) = \log(f(\mathbf{w}|\mathcal{D})) + \frac{N}{2}\log(2\pi) + \frac{1}{2}\log(|\mathbf{A}^{-1}|)$. From Equation 1, $\log(f(\mathbf{w}|\mathcal{D}) = \log(p(\mathbf{y}|\mathbf{X})) + \log(p(\mathbf{w}))$ so substituting back into the model evidence, noting the correlation with Equation 2 and that the terms in π from the prior and the determinant cancel out:

$$\log(p(\mathbf{X})) = \mathcal{L}(\mathbf{w}) + \frac{1}{2}\log(|\mathbf{A}^{-1}|) \tag{6}$$

Computation A higher model evidence indicated a better fit, so it was used to distinguish between parameters. This requires calculating $|\mathbf{A}^{-1}|$ - the introduced numerical errors would often result in a determinant of $\pm\infty$. Two steps were taken to get around this:

- As $|\mathbf{A}^{-1}| = \frac{1}{|\mathbf{A}|}$, it follows $\log(|\mathbf{A}^{-1}|) = -\log(|\mathbf{A}|)$. Skipping the inversion step eliminates associated numerical errors.
- $|\mathbf{A}|$ was calculated via Cholesky decomposition, which decomposes $\mathbf{A}$ into a lower triangular matrix $\mathbf{L}$ such that $\mathbf{A} = \mathbf{L}\mathbf{L}^\intercal$. As it is lower triangular, $|\mathbf{L}|$ is the product of the diagonal elements of $\mathbf{L}$ - which, after a logarithm, is an easy sum. Conveniently, $|\mathbf{A}| = |\mathbf{L}|^2$

Therefore, $\log(\mathbf{A}^{-1}) = -2\sum_i \log(L_{ii})$. This is a much more trivial calculation which doesn't run into computational difficulties faced by the direct computation of $|\mathbf{A}^{-1}|$. The implementation of this is shown in Listings 2 below.

```
1  def model_evidence(x, y, w, hess, prior_variance):
2      # Compute likelihood (f(x0))
3      ev = -posterior(w, prior_variance, x, y)
4
5      # Use Cholesky decomposition of Hessian to avoid numerical errors
6      chol = np.linalg.cholesky(hess)
7      ev -= np.sum(np.log(np.diag(chol)))
8      return ev
```

Listing 2: Function for computing the model evidence (normalising constant X)

2.1.3 Predictive Distribution

The predictive distribution is given by marginalising likelihood with respect to the posterior:

$$p(y^\star = 1|\mathbf{x}^\star, \mathcal{D}) = \int p(y^\star = 1|\mathbf{x}^\star, \mathbf{w}) p(\mathbf{w}|\mathcal{D}) d\mathbf{w}$$

By approximating $p(\mathbf{w}|\mathcal{D}) \approx \mathcal{N}(\mathbf{w}|\mathbf{w}_{\text{MAP}}, \mathbf{A}^{-1})$ and setting $a := \mathbf{w}^\intercal \mathbf{x}^\star$, this intractable integral estimated as:

$$p(y^\star = 1|\mathbf{x}^\star, \mathcal{D}) \approx \int \sigma(\mathbf{w}^\intercal \mathbf{x}^\star) q(\mathbf{w}) \mathrm{d}\mathbf{w} = \int \sigma(a) p(a) \mathrm{d}a \tag{7}$$

It can be shown that, as $q(\mathbf{w})$ is linear Gaussian, so is a. The mean and covariance can thus be established analytically as done in [1], the results of which are given below:

$$\mu_a = \mathbf{w}_{\text{MAP}}^\intercal \mathbf{x}^\star$$

$$\sigma_a^2 = \mathbf{x}^{\star\intercal}\mathbf{A}^{-1}\mathbf{x}^\star$$

Finally, the sigmoid function is approximated by a probit function scaled such that they have the same slope at the origin:

$$\sigma(a) \approx \phi(a\lambda) \qquad \lambda^2 := \frac{\pi}{8}$$

This is useful as the convolution of a probit function with a Gaussian is simply another probit function (Equation 8):

$$\begin{aligned}\int \sigma(a)\mathcal{N}(a|\mu_a, \sigma_a^2)\mathrm{d}a &\approx \int \phi(\lambda a)\mathcal{N}(a|\mu_a, \sigma_a^2)\mathrm{d}a \\ &= \phi\left(\frac{\mu_a}{(\lambda^{-2} + \sigma_a^2)^{1/2}}\right) \qquad (8) \\ &\approx \sigma\left(\frac{\mu_a}{(1 + \lambda^2\sigma_a^2)^{1/2}}\right) \qquad (9)\end{aligned}$$

Equation 9 is the final form of the predictive distribution, with μ_a and σ_a derived from the training data, the Hessian and the MAP weights. The implementation of the predictive distribution is shown in Listings 3.

```
def bayesian_prediction(cov, x, w):
    # Bias inputs
    x_tilde = np.concatenate((x, np.ones((x.shape[0], 1))), 1)

    # Calculate mean and variance of Laplace approximation
    ## Note: Covariance is the inverse of the Hessian
    mean = x_tilde @ w
    var_a = np.diag(x_tilde @ (cov @ x_tilde.T))

    # Approximate logistic with probit for integral over posterior.
    ## This gives another probit. Approximate that probit with logistic.
    k_probit = (1 + 0.125 * np.pi * var_a) ** -0.5
    return logistic(k_probit * mean)
```

Listing 3: Function for computing the Bayes prediction, using the Laplace approximation

It is worth noting that, both the predictive distribution and the MAP weights will predict the same $y^\star$ for a given $\mathbf{x}^\star$ as $p(y^\star = 1) = 0.5$ when $\mathbf{w}^\intercal\mathbf{x}^\star = 0$ for both solutions.

3 Model performance

The shuffling functionality used in the short lab to further randomise data sets was disabled for this FTR in order to simplify the number of variables.

3.1 ML Optimisation

It was desired to compare MAP/fully Bayesian solutions to the original ML solution. When the maximum likelihood distributions were passed through SciPy's optimiser, it gave results different to the short lab.
Because there is no prior here, when training, the weights can essentially keep growing while

keeping the same proportions i.e. $\mathbf{w} \to k\mathbf{w}$. This means contours of $\sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})$ have the same shape but values of $\mathbf{w}^\intercal \mathbf{x}$ get pushed to very large negative or positive values, forcing the outputs of $\sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})$ to become 1 or 0 - a significantly overfitted model. This behaviour is illustrated in Figure 1 where the SciPy optimised ML distribution is shown. The probability contours remain roughly the same shape regardless of number of iterations, but they collapse around the decision boundary as iterations increase. This also caused issues when calculating the log-likelihood as $\log(1 - \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)}))$ for $\sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)}) = 1$ would fail.
Though this behaviour was anticipated it did cause some difficulty in terms of choosing an arbitrary amount to train the maximum likelihood classifier. It is suspected that the SciPy optimiser is much more powerful than the iterative method implemented in the short lab, which likely converged on some slightly sub-optimal minimum - the leftmost solution in Figure 1 will have a higher average log-likelihood as the model is more 'certain' of its prediction, so it is a technically more optimal solution based only on the cost function, resulting in the behaviour in Figure 1.
The difficulties faced here arguably illustrate the benefits of a Bayesian approach more than any of the results from here onwards.

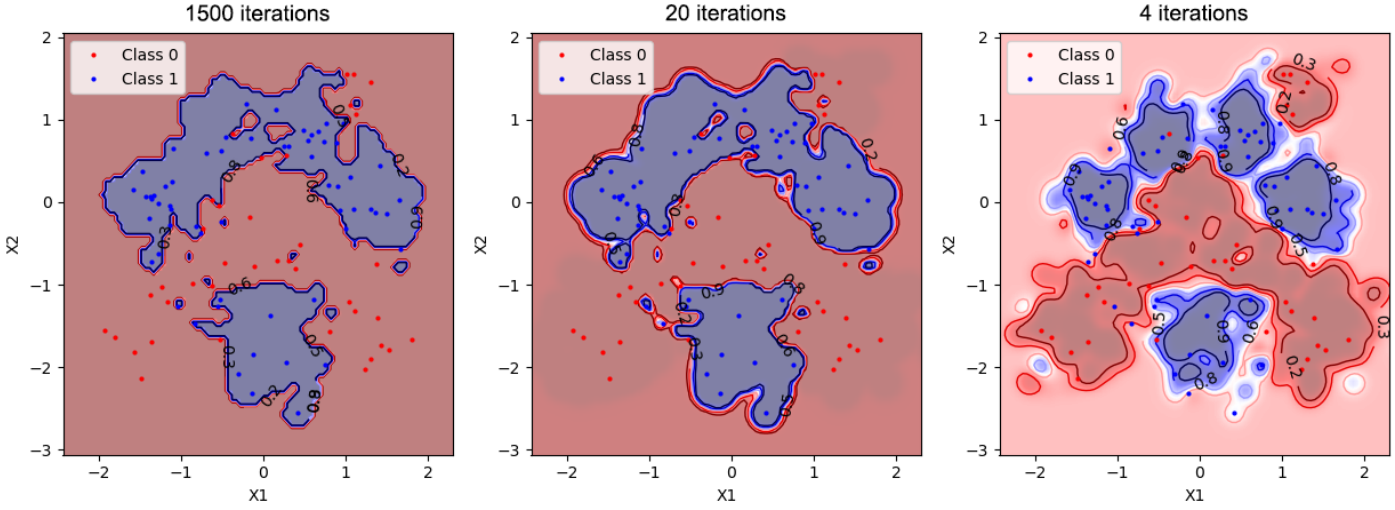


Figure 1: Max likelihood contours with different numbers of iterations, for $\sigma_0 = 1$, $l = 0.1$

3.2 Performance on Base Case

The base case analysed had prior variance, $\sigma_0 = 1$ and RBF variance, $l = 0.1$. The fully Bayesian solution was compared to the MAP solution *and* the ML solution. Contours for these are shown in Figure 2, and final test and train log-likelihoods are shown in Table 1. Tables 2 and 3 show the confusion matrices for training and test data respectively of the three solutions.

Table 1: Average log-likelihoods, $\sigma_0 = 1$, $l = 0.1$

	ML	MAP	Bayes
Train	-0.060	-0.220	-0.260
Test	-0.551	-0.334	-0.353

Table 2: Confusion matrices for **training data** of ML, MAP and Bayes solution for $\sigma_0 = 0.1$ and $l = 0.1$

ML

Ground truth	Prediction 0	Prediction 1
$y = 0$	0.98	0.02
$y = 1$	0.02	0.98

MAP

Ground truth	Prediction 0	Prediction 1
$y = 0$	0.95	0.05
$y = 1$	0.05	0.95

Bayes

Ground truth	Prediction 0	Prediction 1
$y = 0$	0.95	0.05
$y = 0$	0.05	0.95

Table 3: Confusion matrices for **testing data** of ML, MAP and Bayes solution for $\sigma_0 = 0.1$ and $l = 0.1$

ML

Ground truth	Prediction 0	Prediction 1
$y = 0$	0.84	0.16
$y = 1$	0.07	0.93

MAP

Ground truth	Prediction 0	Prediction 1
$y = 0$	0.91	0.09
$y = 1$	0.12	0.88

Bayes

Ground truth	Prediction 0	Prediction 1
$y = 0$	0.91	0.09
$y = 0$	0.12	0.88

Overfitting The ML tendency to overfit (and the MAP solution's better performance here) is evident in Tables 2 and 3; it has a higher prediction accuracy than MAP/Bayes for the training data but then has a lower prediction accuracy for the testing data. This is also seen in the log likelihoods; the ML log-likelihood is significantly higher for training data but significantly lower for testing data.

Prediction accuracy Aside from performing better than the ML solution, it can be seen in Tables 2 and 3 the MAP and Bayes solutions have the exact same prediction accuracies. They are directly compared in the style of a confusion matrix in Table 4 where it can be seen they predict exactly the same thing for all testing/training points. This was expected; the decision boundary is not effected for the Gaussian case because the modal weights around which the Laplace approximation is fitted is $\mathbf{w}_{\text{MAP}}$. This is reflected in the contours in Figure 2 - the contours for $p(y^{(n)} = 1) = 0.5$ (the decision boundary) are exactly the same.

Table 4: MAP vs. fully Bayesian predictions

Training

Bayes	MAP $y_M = 0$	MAP $y_M = 1$
$y_B = 0$	1.00	0.00
$y_B = 1$	0.00	1.00

Testing

Bayes	MAP $y_M = 0$	MAP $y_M = 1$
$y = 0$	1.00	0.00
$y = 1$	0.00	1.00

The predictive distribution integrates over all possible solutions of $\mathbf{w}$, weighting the resultant prediction with the posterior likelihood of that particular $\mathbf{w}$, so it is expected that there would be more uncertainty in the predictions as the uncertainty in $\mathbf{w}$ carries forward. This is seen in the contours as shrinkage, which translates to a smoother peak in the distribution. It can also be seen in Table 1 - the log likelihood of the MAP solution is higher than the Bayes solution for both training and test data.

Contours Considering the form of the Hessian (equation 5), it can be seen that all elements of $\mathbf{A}$ are positive. The term due to the prior will be entirely positive or zero, and as $0 < \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)}) < 1$, then $\sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})(1 - \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)}))$ will also be positive. Furthermore, the output

of the rbf function is always positive so $\mathbf{x}^{(n)}\mathbf{x}^{(n)\intercal}$ will have only positive elements. Therefore, as the amount of training data increases, the Hessian grows more positive. As the Hessian grows, $\mathbf{A}^{-1}$ becomes smaller and so does $\sigma_a^2 = \mathbf{x}^{\star\intercal}\mathbf{A}^{-1}\mathbf{x}^{\star}$. As a result, $\lambda^2\sigma_a^2$ in the Bayesian prediction (equation 8) approaches 0, and it tends towards $\sigma(\mu_a) = \sigma(\mathbf{w}_{\text{MAP}}\mathbf{x})$ - i.e. the MAP prediction. This is expected - as more data points are added the prior becomes less and less significant. This explains why, when using 75% of the data for training the Bayesian contours look very similar to the MAP contours.

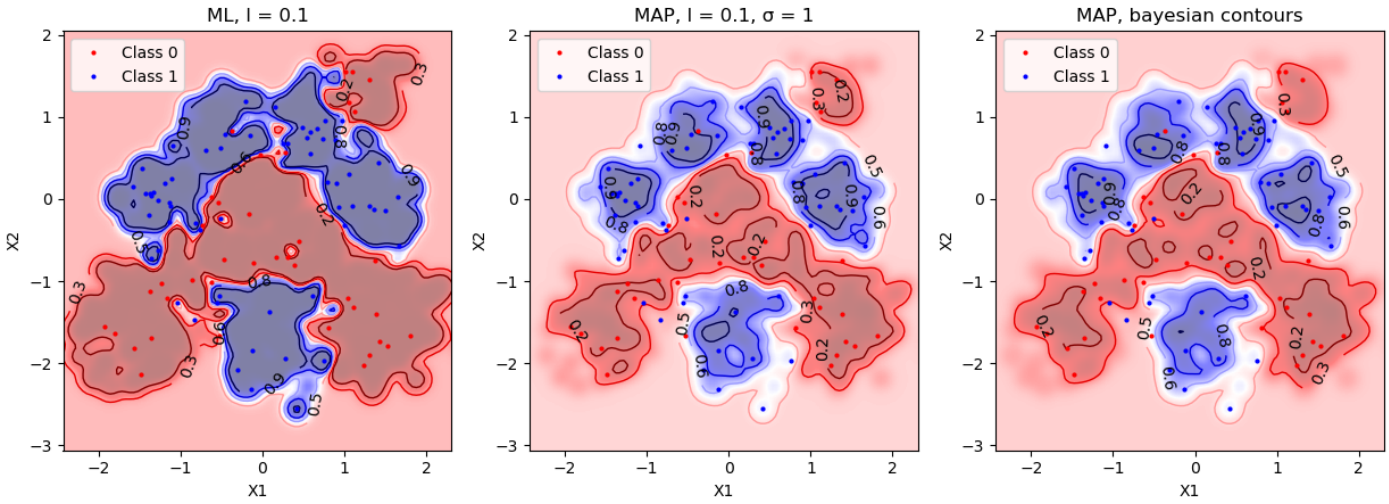


Figure 2: Contour plots for ML, MAP and fully Bayesian solution for $\sigma_0 = 1$, $l = 0.1$

If only 1% of the data is used for training, the effect of the prior is more obvious, as shown in Figure 3. The overfitting of the ML solution is very evident here, as is the increased uncertainty of the fully bayesian contours.

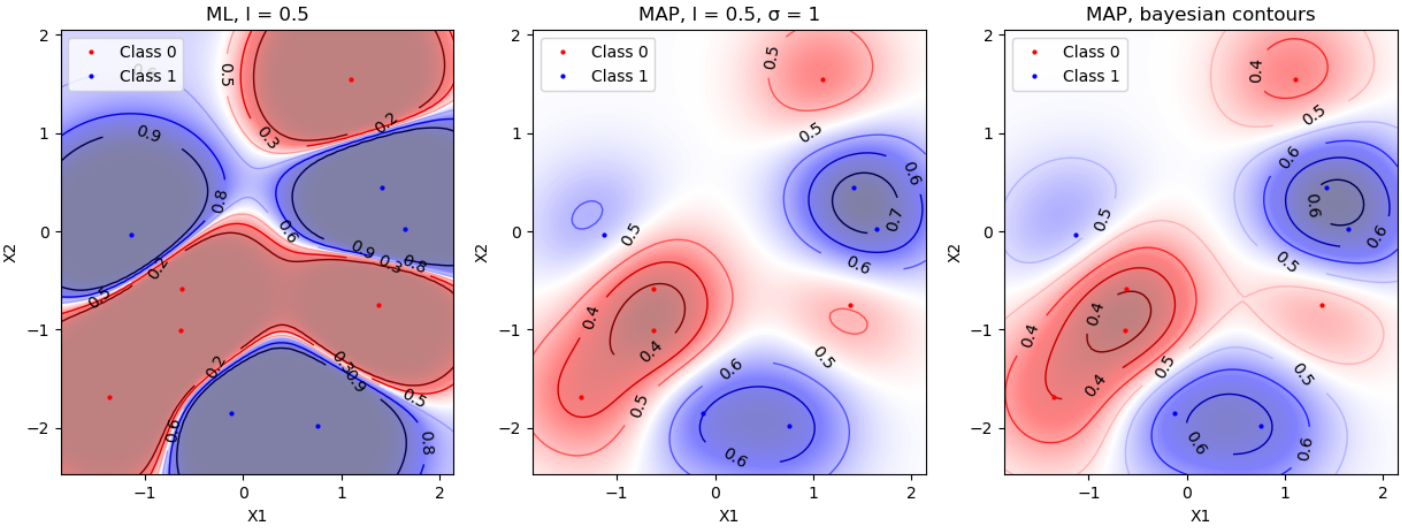
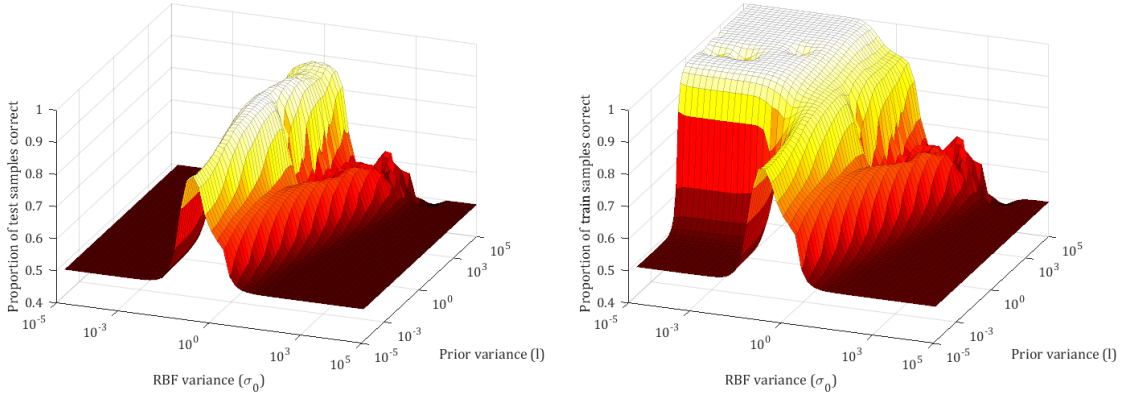


Figure 3: Contour plots for ML, MAP and fully Bayesian solutions using only 10 training points

3.3 Parameter optimisation

3.3.1 Grid search

A ten by ten grid search was performed, varying the parameters logarithmically in the range $10^{-4} < \sigma_0 < 10^4$ and $10^{-4} < l < 10^4$. It was varied logarithmically as both terms appear inside exponentials so logarithmic variation produces a more uniformly varying output.
According to this 10x10 grid search, the best parameters were $\sigma_0 = 6.31$ and $l = 1$ with a predictive accuracy of 0.9 and a model evidence $\log(X) = -188.4$. A more comprehensive 50 x 50 grid search was also performed in order to plot model performance on a 3D surface graph, for which [2] was used.



(a) Testing data (b) Training data

Figure 4: Prediction accuracy with varying parameters, 50 x 50 grid search

Firstly, most feasible solutions lie within the $0.1 < \sigma_0, l < 10$ so a more sensible 10 x 10 grid search could have been done.
It can be seen from Figures 4 and 5 that there are a large number of solutions for $\sigma_0^2 \approx 1$ which give roughly equally high prediction accuracies. In this region, the shape of the training data accuracy closely mimics the testing data, if slightly higher. The region of very low RBF variance is where the model overfits significantly as discussed in the short lab, but with very small prior variances the model can't even predict training data correctly. This is likely because the contours are so tight around the training points that any adjustment from such a strong prior completely ruins predictions.

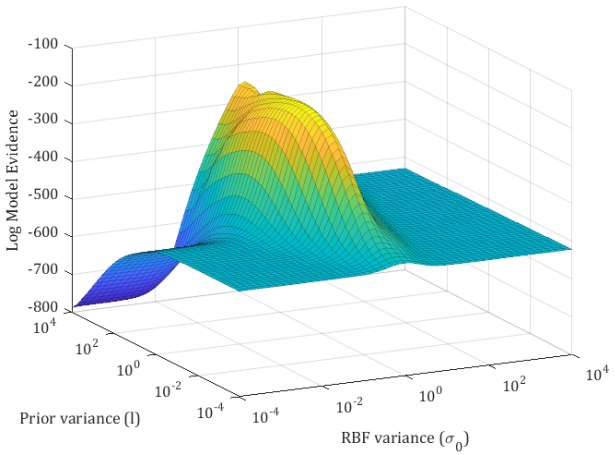


Figure 5: Model evidence with varying parameters, 50 x 50 grid search

For very large σ_0, the $\frac{1}{\sigma_0^2}$ term becomes negligible and the cost function becomes that of the ML problem, so $\mathbf{w}_{\text{MAP}} \to \mathbf{w}_{\text{ML}}$. This is seen in Figure 6. For such a large σ_0^2 the prior is effectively uniform. It doesn't favour any $\mathbf{w}$ over another, so the Bayesian integral spreads the distribution out very significantly. Attempts to optimise σ_0 and l automatically using

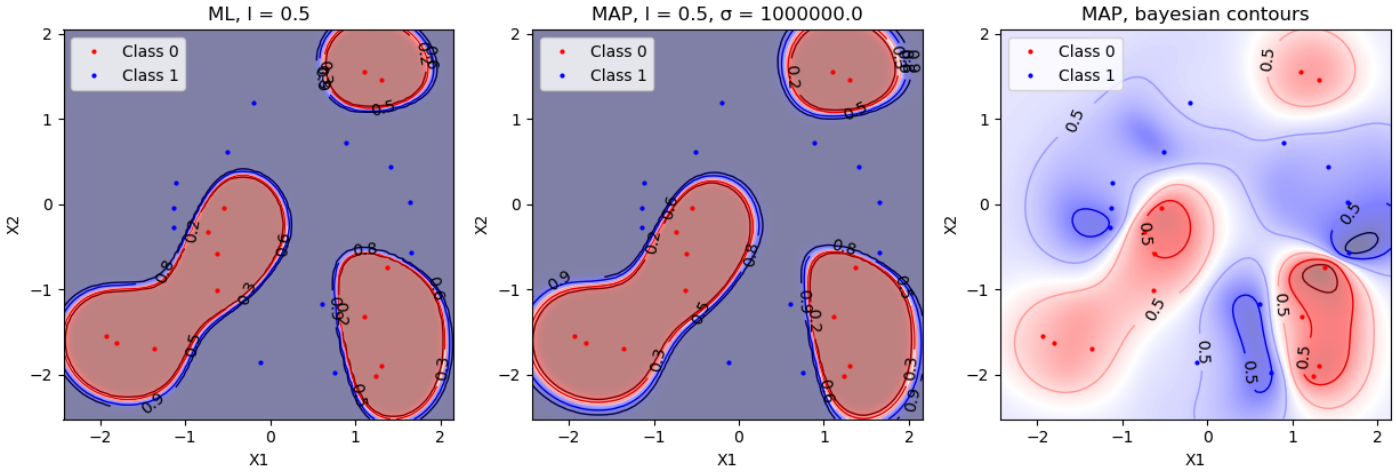


Figure 6: The MAP solution collapses into the ML solution at large σ_0^2, but the Bayesian approach to prediction reduces overfitting.

SciPy's optimiser did not converge on the most optimal solutions. This is likely due to the very small gradients at the $\sigma_0 = 1$ ridge where optimal values lie.

3.3.2 Best parameters

The best parameters from the 50 x 50 grid search were $l = 0.479$ and $\sigma_0 = 0.692$, giving a predictive accuracy of 0.91 and a model evidence $\log(p(\mathbf{X})) = -175.81$. Though this is an improvement from the $\sigma_0 = 0.1, l = 1$ base case, as noted, there are many parameter combinations with very similar performances. Furthermore, accuracies of up to 0.924 were observed so the question remains whether its worth optimising based on that metric instead.

Table 5: Final MAP predictions for $l = 0.479$ and $\sigma_0 = 0.692$

	Training			*Testing*	
	Prediction			Prediction	
Ground truth	$\hat{y} = 0$	$\hat{y} = 1$	Ground truth	$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$	0.94	0.06	$y = 0$	0.90	0.10
$y = 1$	0.07	0.93	$y = 1$	0.08	0.92

Table 6: Best case average log-likelihoods, $l = 0.479$ and $\sigma_0 = 0.692$

	MAP	BAYES
Train	-0.175	-0.184
Test	-0.218	-0.220

4 Conclusions

The work carried out for the full technical report has significantly expanded on the short lab and in particular shown the following key points:

1. Bayesian classification is a fairly robust way of tackling overfitting, providing accuracies above that of the maximum likelihood classification approach
    - It does however come at the cost of increased computational complexity and introduction of more parameters to tune.
2. The Laplace approximation was developed and shown to estimate the posterior distribution to a highly acceptable accuracy given the computation savings and avoided complexity.
3. The normalising constant of the Laplace approximation provides a computationally inexpensive way to assess the model in order to optimise hyperparameters

Further development could be done in automating the hyperparameter optimisation and exploring the use of different priors/basis functions.

References

[1] Pattern Recognition and Machine Learning
Bishop, C. M., *New York: Springer*, 2006. Print

[2] Surface Fitting using Gridfit
D'Errico, J., *MATLAB File Exchange*, 2006

A Short Lab Report

This was submitted on Moodle on 14/02/2018

A.1 Introduction

Classification is a form of supervised learning whereby a set of data is assigned one of finite discrete labels - for example, given an image, label it as either a dog or a cat. This finds huge usage and application in computer vision, finance, healthcare, modelling sciences and more. One such algorithm, *logistic regression*, discretises the prediction of a linear regression model (e.g. if there are labels 0 and 1, a linear regression model may predict 0.43, 0.59 and 1.03 - logistic regression bridges the gap, returning labels 0, 1 and 1).
In this coursework, logistic regression was explored with and without the use of radial basis functions to reparametrise the model.

A.2 Methods developed

A.2.1 Model analysis

The data is modelled according to $y^{(n)} = \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})$, where $y^{(n)}$ and $\mathbf{x}^{(n)}$ are the class and parameters respectively. The likelihood $P(\mathbf{y}|\mathbf{X}, \mathbf{w})$ and log-likelihoods ($\mathcal{L}$) are given by:

$$P(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} P(y^{(n)}|\mathbf{x}^{(n)}) = \prod_{n=1}^{N} \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})^{y^{(n)}} (1 - \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)}))^{1-y^{(n)}} \tag{1}$$

$$\mathcal{L}(\mathbf{w}) = \log(P(\mathbf{y}|\mathbf{X},\mathbf{w})) = \sum_{n=1}^{N} y^{(n)} \log(\sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})) + (1 - y^{(n)}) \log(1 - \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})) \tag{2}$$

Considering the gradient:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \frac{\partial \mathcal{L}}{\partial \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})} \frac{\partial \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})}{\partial \mathbf{w}^\intercal \mathbf{x}^{(n)}} \frac{\partial \mathbf{w}^\intercal \mathbf{x}^{(n)}}{\partial \mathbf{w}} \\
&= \sum_{n=1}^{N} y^{(n)} \times \frac{1}{\sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})} \times \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})(1 - \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})) \times \mathbf{x}^{(n)} \\
&\qquad + (1 - y^{(n)}) \times \frac{-1}{1 - \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})} \times \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})(1 - \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})) \times \mathbf{x}^{(n)} \\
&= \sum_{n=1}^{N} y^{(n)}(1 - \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)}))\mathbf{x}^{(n)} - (1 - y^{(n)})\sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})\mathbf{x}^{(n)} \\
&= \sum_{n=1}^{N} y^{(n)} \mathbf{x}^{(n)} - \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})\mathbf{x}^{(n)} \\
&= \sum_{n=1}^{N} (y^{(n)} - \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)}))\mathbf{x}^{(n)}
\end{aligned} \tag{3}$$

A.2.2 Implementation

The pseudocode in Algorithm 1 was implemented, using a custom iterator which generates randomised batches of a specified batch size. This was likely not necessary for this exercise as the dataset is fairly small and thus it is not wasteful to update parameters at the end of

each epoch as opposed to at the end of each minibatch.

Algorithm 1: Parameter estimation pseudocode

Data: Input data $\mathbf{X}$, corresponding classes $\mathbf{y}$, training rate η
Result: Parameter $\mathbf{w}$
Initialise $\mathbf{w}_{\text{new}} \leftarrow \mathcal{N}(0,1)$, length $= N_{params} + 1$
while $|\mathbf{w}_{old} - \mathbf{w}_{new}| < threshold$ ***AND*** $n\_epochs < max\_epochs$ **do**
  Go through entire epoch
  for batch *in* randomised batches **do**
    error $= \text{transpose}(\mathbf{y}_{\text{batch}} - \text{sigmoid}(\mathbf{X}_{\text{batch}} \times \mathbf{w}_{\text{new}})) \times \mathbf{X}_{\text{batch}}$ ▷ *Column vectors*
    $\mathbf{w}_{\text{old}} \leftarrow \mathbf{w}_{\text{new}}$
    $\mathbf{w}_{\text{new}} \leftarrow \mathbf{w}_{\text{new}} + \eta \times \text{error}$

Note: Could use the change in log likelihood at every step and check if that is below a threshold to measure convergence

Dataset The dataset is visualised in Figure 7. It was chosen to use 25% of the data as testing data and 75% as training data. It is important these are kept separate as if the testing data influences the model parameters in any way, the generalisability of the model cannot be tested.
Random indices were chosen for the dataset - this prevents the model being optimised specifically for training on the first 75% of the dataset and testing on the last 25%.

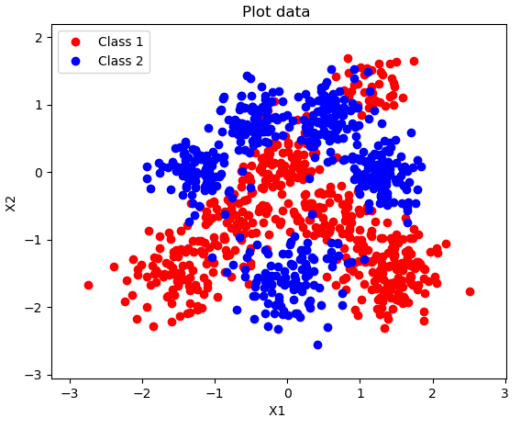
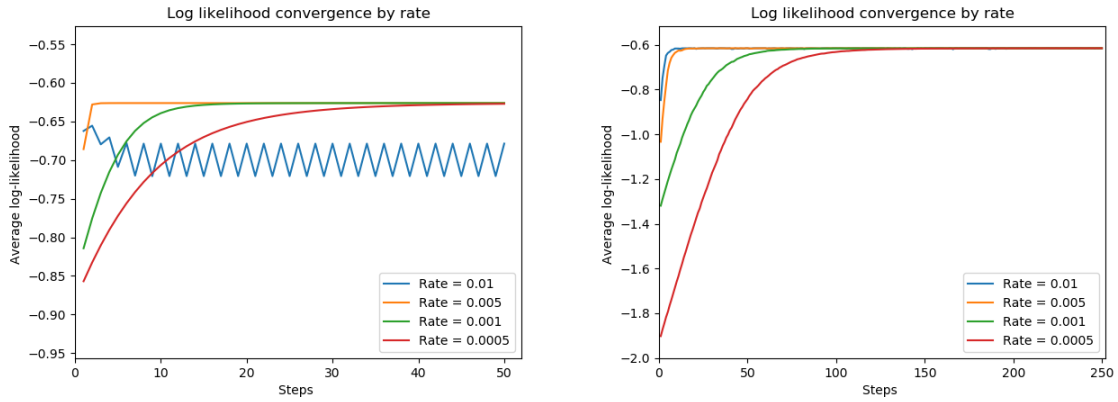


Figure 7: The dataset used.

Convergence criteria It was implemented such that, should the log-likelihood of the training data change by an amount below the threshold, it will stop training and assume converged. Ideally a validation data set should be used such that the log-likelihood of that can be used to assess convergence.

Training rate In general, a higher training rate resulted in less training being required to converge at the risk of potential oscillation. This is visualised in Figure 8a. Training rate was thus chosen on a trial-and-error basis; the easiest way to pick parameters was to observe the evolution of log-likelihood and change number of epochs/training rate accordingly. Parameter optimisation can however be automated. Interestingly, use of minibatches allowed for higher training rates without oscillation - see Figure 8b. This is because the smaller minibatch updates gives a smaller effective training rate per update, increasing stability.



(a) No minibatches (b) Minibatch size = 50

Figure 8: Effect of training rate on convergence of linear logistic classifier

A.2.3 Models

The model $y^{(n)} = \sigma(\mathbf{w}^\intercal \mathbf{x}^{(n)})$ can have it's inputs reparametrised by the use of basis functions. These transform inputs $[1, X_1, X_2]$ into $[1, \phi_1(X_1, X_2), \phi_2(X_1, X_2), ..., \phi_n(X_1, X_2)]$. Two versions of the model were used - a linear logistic classifier, with no basis function (i.e. $\phi_1(\mathbf{x}) = x_1$ and $\phi_2(\mathbf{x}) = x_2$), and one with radial basis functions (equation below).

$$\phi_m(\mathbf{x}^{(n)}) = \exp\Big(-\frac{1}{2l^2} |\mathbf{x}^{(m)} - \mathbf{x}^{(n)}|_2^2 \Big)$$

The radial basis functions reinterpret inputs in terms of (unscaled) gaussian functions. There is one basis function for each training data point, and $\phi_m(\mathbf{x}^{(n)})$ gives the (unscaled) probability that a gaussian, centred at point $\mathbf{x}^{(m)}$, would generate a point with coordinates $\mathbf{x}^{(n)}$. This introduces another parameter - l, the variance of the gaussian functions. The effect and interpretation of l are discussed in Section A.5

A.3 Model performance

A.4 Linear Classifier

Based alone off the visualisation of the data (Figure 7), it was not expected a linear classifier would work well. Though the two classes are fairly distinct, no single linear boundary could be drawn that separates them well.
The confusion matrices are shown in Table 7. As expected, the model does not do particularly well, misclassifying approximately a quarter of testing and training cases.

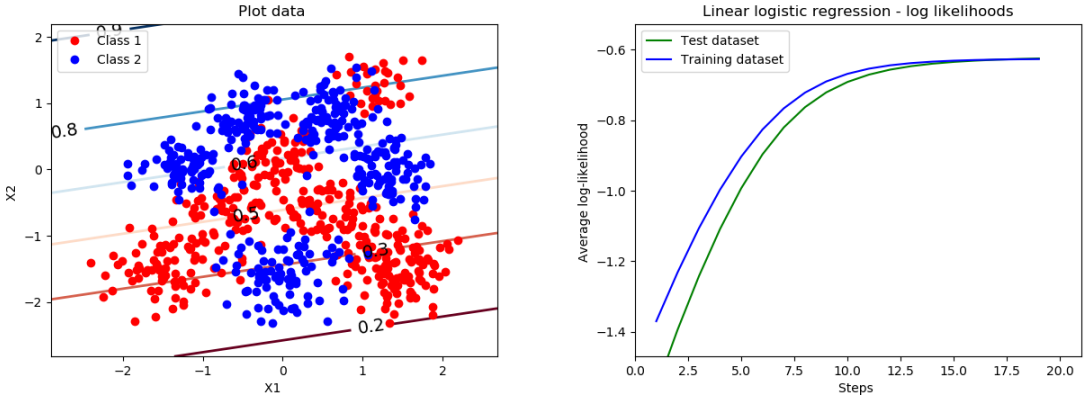
Table 7: Linear logistic classifier confusion matrices

	Training prediction	
Ground truth	$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$	0.74	0.26
$y = 1$	0.29	0.71

	Test prediction	
Ground truth	$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$	0.75	0.25
$y = 1$	0.25	0.75

It can be seen from the contour plot in Figure 9a why the model performs badly - as suspected, there is no linear boundary which adequately describes the data.
In terms of convergence, the model typically converged after about 20 epochs of data. The evolution of log-likelihood is shown in Figure 9b. In this case, the average log-likelihoods

were -0.624 and -0.631 for training and testing respectively. Due to the randomised selection of both the training data and the mini-batches, this varied slightly between runs but was broadly similar.



(a) Probability contours of linear logistic classifier (b) Log-likelihood evolution with training steps

Figure 9: Performance of linear logistic classifier

A.5 Radial Classifier

By producing more flexible boundaries, the radial classifier was generally able to fit the data better. The most important thing then becomes the effect of the variance parameter l.
The radii of the basis functions are dictated by l. A smaller l results in basis functions which have a lower 'probability' of generating further away points. This results in probability contours which are closer to the training data. This can be seen in Figure 10 - the smoothest contours/largest boundaries are given by the larger l.

A.5.1 Model accuracy

Confusion matrices for the different l values, for training and testing data, are shown in Table 9 & 8 respectively. It can be seen that decreasing l results in a better fit to the training data but a poorer model accuracy for testing data. The reasoning behind this is discussed in Section A.5.2 below.
In general, with good values of l the model was able to achieve results significantly better than the linear classifier (Up to 95/86% accuracy vs. 75/75%). The final log likelihoods shown in Table 10 show a higher log-likelihood for most of the RBF cases, further indicating that it is better able to fit the data.

Table 8: RBF Logistic classifier testing confusion matrices

$l = 1$

Ground truth	Prediction 0	Prediction 1
$y = 0$	0.83	0.17
$y = 1$	0.27	0.73

$l = 0.1$

Ground truth	Prediction 0	Prediction 1
$y = 0$	0.95	0.05
$y = 1$	0.14	0.86

$l = 0.01$

Ground truth	Prediction 0	Prediction 1
$y = 0$	0.98	0.02
$y = 0$	0.91	0.09

Table 9: RBF Logistic classifier training confusion matrices

$l = 1$

Ground truth	Prediction 0	Prediction 1
$y = 0$	0.86	0.14
$y = 1$	0.14	0.86

$l = 0.1$

Ground truth	Prediction 0	Prediction 1
$y = 0$	0.94	0.06
$y = 1$	0.08	0.92

$l = 0.01$

Ground truth	Prediction 0	Prediction 1
$y = 0$	1.00	0.00
$y = 0$	0.00	1.00

Table 10: Final log-likelihoods

	RBF: $l = 1$	RBF: $l = 0.1$	RBF:$l = 0.01$	Linear
Test	-0.44	-0.25	-0.67	-0.63
Training	-0.36	-0.20	-0.07	-0.63

A.5.2 Generalisability

Smaller ls overfit - they tune the contours to the training data better, giving better predictions for the training set while losing generalisability. This is evident in Tables 9 and 8 - for smaller l, the training confusion matrix becomes $\mathbf{I}$; i.e. perfect prediction. Figure 10c shows the issue - prediction boundaries are only well defined close to training data points, so it only predicts the correct class for a test point if is very close to a training point of the same class.

$l = 0.01$ case

The testing data confusion matrix for $l = 0.01$ showed some interesting behaviour - it predicted 0 for almost all the data points. However, due to the random shuffling at the dataset creation phase, this sometimes flipped, with the model predicting predominately 1s.

This is likely due to the very localised basis functions - they effectively only have significant values very close to the data point. Far away, the probability that any data point is generated by a gaussian at each of the training data points is going to be roughly the same (very small) number for all of them, as the exponent is more controlled by the $\frac{1}{l^2}$ term than the distance (l_2 norm). The probability of class 0 is just the sum of roughly equivalent far-field probabilities from all the training class 0 points. As a result, if there are more 0s in the training dataset it will predominantly pick 0s as at any point not near to a training data point, the class-0 Gaussians have a stronger effect. What the model predicts is less a function of the geometry of the training points and more dictated by which class is more present.

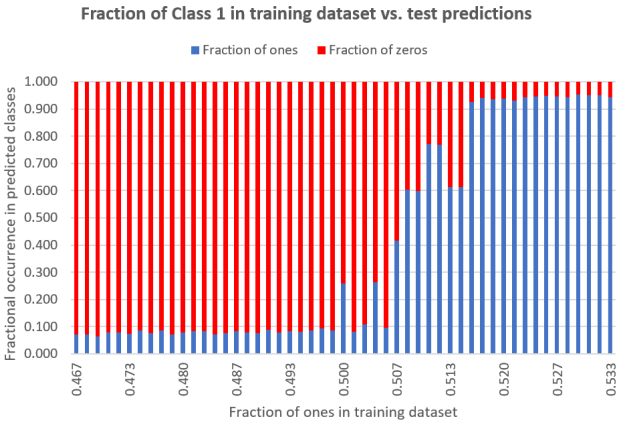
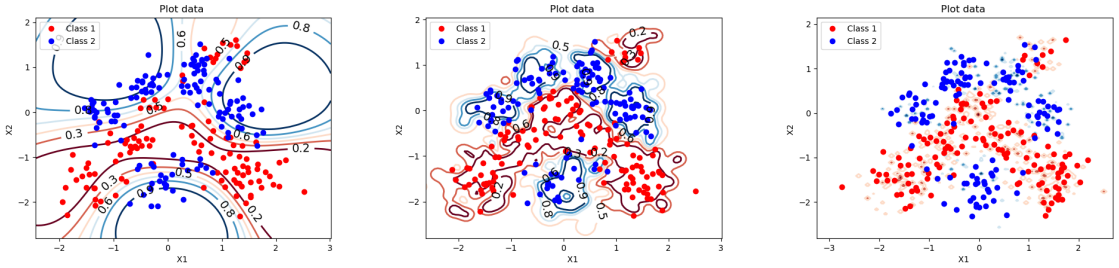


Figure 11: Fraction of 1s/0s for different compositions of dataset

In order to test this, the dataset was forced to have specific proportions of 0s/1s. The fraction of 0s/1s predicted by the model was then recorded - for a perfect model it should be roughly half each as the dataset has 494 ones and 506 0s. This was repeated multiple times and averaged, and is shown in Figure 11.

As expected, when the fraction of 1s in the training set was below 0.5, it mostly predicted 0s.



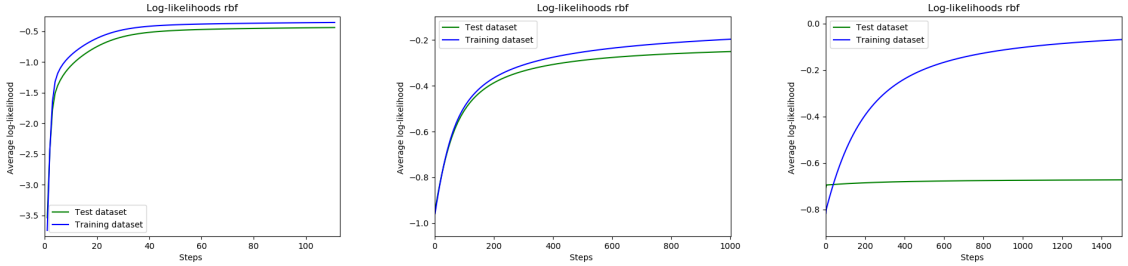
(a) $l = 1$ (b) $l = 0.1$ (c) $l = 0.01$

Figure 10: RBF Logistic classifier: Contour maps, with testing data points

Near 0.5, the arrangement of points plays a more significant role and it's less dictated by the quantity of each class. However, above 0.51 it predicts primarily 1s for all the testing points.

A.5.3 Training rates

Lower l required higher training rates in order to achieve convergence in a given number of epochs. The training rates are shown in Figure 12 alongside the evolution of log likelihood.Smaller l requires more training to converge as the contours can become more specific/complex than the larger l cases allow, so with more training it can still improve its log-likelihood for the training case.



(a) $l = 1, \eta = 1 \times 10^{-4}$ (b) $l = 0.1, \eta = 1 \times 10^{-3}$ (c) $l = 0.01, \eta = 1 \times 10^{-2}$

Figure 12: RBF Logistic classifier: Log-likelihood evolution

A.6 Computational complexity

Increased dimensionality meant the RBF classifier took not only noticeably longer to train for each epoch, but it also required more training steps to converge (500+ vs 20 epochs).

A.7 Conclusions

1. Training rates for logistic classification must be tuned bearing in mind the tradeoff between slow convergence and potential instability
2. The use of radial basis functions can make a logistic classifier more flexible & accurate, at the expense of increased computational complexity
3. The l parameter of the radial basis functions controls the generalisability of the model - with a small enough l, it perfectly predicts training data and acts as a low-accuracy majority system for unseen data.