University of Cambridge

Engineering Tripos Part IIA

Girton College

3F8 Full Technical Report
Bayesian Binary Classifier

Clinton N. Igwegbu (cni22)

March 19, 2019

1 Introduction

The short report involved implementing a logistic classifier with the maximum-likelihood solution. Noting that multiple decision boundaries can be used to classify data, this long report implements a Bayesian classifier, where each boundary is assigned a prior probability. A maximum-a-posteriori (MAP) implementation makes predictions using the parameters that maximise the posterior distribution. The full Bayesian implementation effectively performs a weighted sum of the predictions from different parameters. Both the MAP and full Bayesian approaches are investigated in this long report.

2 Theory

2.1 Model

When predicting the classification ($y^* = 1$ or 0) of a new data point x^*, the probability that the new point belongs to class 1 is taken to be the following expression:

$$p(y^* = 1|x^*, \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x}^*) \tag{1}$$

Note how the above expression makes a prediction of the new point's class based on the value of the parameter, $\mathbf{w}$, used to predict it. In the short lab, $\mathbf{w}_{ML}$, the max-likelihood estimate $\mathbf{w}$ was used to make this prediction. However it must be noted that in multi-dimensional data sets several values of $\mathbf{w}$ can successfully divide the two classes.

By assigning a prior distribution on w, multiple values can be tried, each with a different associated probability density. The likelihood that a particular data point, $\mathbf{x}$,

belongs to class 1 given a decision boundary, $\mathbf{w}$, is the same as that seen in the short report i.e. $\sigma(\mathbf{w}^T\ \mathbf{x})$. The posterior distribution is proportional to the prior times the likelihood:

Prior Distribution

$$p(\mathbf{w}) = \frac{1}{\sqrt{(2\pi\sigma_0^2)^M}} \exp\left(-\frac{1}{2\pi\sigma_0^2}\mathbf{w}^T\mathbf{w}\right) \tag{2}$$

Where M is the no. of parameters and σ_0 is the variance of each parameter.

Posterior Distribution

$$p(\mathbf{w}|D) = \frac{1}{Z}\, p(w)p(D|w) \tag{3}$$

Where D is the training data i.e (X, y) and Z is the normalising constant i.e. the evidence, p(D).

Evidence

$$p(D) = Z = \int p(w)p(D|w)\,\mathrm{d}w \tag{4}$$

This long report first makes predictions using w_{MAP}, the value of $\mathbf{w}$ that maximises the posterior, instead of $\mathbf{w}_{ML}$.

The report then follows a different approach. Equation 1 tells us that each possible value of $\mathbf{w}$ gives a different prediction. A full Bayesian classifier takes a weighted sum of each prediction (where each prediction is a function of $\mathbf{w}$) to form an overall prediction. The weights are given by the posterior on $\mathbf{w}$, and since $\mathbf{w}$ is continuous the sum becomes an integral:

$$p(y^* = 1|x^*, D) = \int p(y^* = 1|x^*, \mathbf{w})p(\mathbf{w}|D)\,\mathrm{d}w \tag{5}$$

$$p(y^* = 1|x^*, D) = \int \sigma(\mathbf{w}^T\mathbf{x}^*)p(\mathbf{w}|D)\,\mathrm{d}w \tag{6}$$

To find the normalising constant, Z, the unnormalised posterior must be integrated (eqn. 4). This integral is not trivial so it would help to approximate the posterior with a Gaussian. A common approach used to do this is known as the Laplace approximation as seen in the next section.

2.2 Laplace Approximation

The Laplace approximation, approximates the posterior as a Gaussian q($\mathbf{w}$) for easy integration. It is found by expanding the Taylor distribution of the unnormalised log posterior, ln f(w), about the MAP solution ($\mathbf{w}_0$) and truncating terms higher than the second order:

$$\ln f(w) \approx \ln f(w_0) + (w - w_0)\nabla f(w)|_{w=w_0} - \frac{1}{2}(w - w_0)^T A(w - w_0) \tag{7}$$

$$\text{where A} = -\nabla\nabla \ln f(w)|_{w=w_0}$$

Noting that the gradient at $w = w_0$ is 0, the second term on the RHS evaluates to 0. Taking exponents it is noted that approximation of f(w), q(w), forms a Gaussian distribution.

$$f(w) \approx f(w_0) \exp(-\frac{1}{2}(w - w_0)^T A(w - w_0)) \tag{8}$$

The normalising constant of a Gaussian may be found by inspection:

$$Z = (2\pi)^{M/2} f(w_0)|A^{-1}| \tag{9}$$

2.3 BIC and Model Evidence

Taking logs of equation 9:

$$\ln p(D) = \ln p(D|w_0) + \ln p(w_0) + \frac{M}{2}\ln(2\pi) + \frac{1}{2}\ln|A| \tag{10}$$

$|A|$ is hard to compute directly for a large matrix so instead the Cholesky decomposition was used. This decomposes A into LL^T where L is a lower triangular matrix. Noting that the determinant of a triangular matrix is the product of its diagonal elements then $|A| = |L|^2$. Hence $\ln|A| = 2\ln|L|$ which is simply 2 times the trace of log L, a very easy sum to compute.

If the prior is assumed to be broad [1]:

$$\ln p(D) \approx -(\frac{1}{2}M \ln N - \ln p(D|w_0)) \tag{11}$$

In plain English, evidence is equal to likelihood minus the number of hyper-parameters times a constant. The expression in parenthesis is known as the BIC (Bayesian Information Criterion). Maximising the likelihood sometimes has the effect of over-fitting the data especially when the model used is too complex. So we aim to maximise the

likelihood while simultaneously minimising the number of hyper-parameters, M, used in the model. Hence, we aim to minimise the BIC. In this long lab, the model on the prior was fixed so the BIC was minimised by maximising the log-likelihood (and therefore the evidence).

In this long report, both approaches above were used to compute the evidence.

2.4 Predictive Distribution

In section 5 of Bishop's book [1] it was noted that equation 6 may be approximated as follows:

$$p(y^* = 1|x^*, D) \approx \sigma(\kappa(\sigma_a^2)\mu_a) \tag{12}$$

$$\text{where } \kappa(x) = \frac{1}{1+\pi^2 x/8},\ \sigma_a = x^{*T} A^{-1} x^* \text{ and } \mu_a = w_{MAP}^T x^*.$$

Note that the decision boundary, i.e. $p(y^* = 1|x^*, D) = 0.5$ is given by $\mu_a = w_{MAP}^T x^* = 0$. Therefore the decision boundary for the full Bayesian approach is the same as that of the MAP approach. Where the two approaches differ is in the non-half probability contours. The classification predictions in this long lab for these two approaches is therefore expected to be identical.

2.4.1 Code for Laplace Approximation and Evidence

In the code below *laplace_parameters* finds the mean and covariance matrix (SN) of q(w).

get_b_map uses scipy.optimize.fmin_l_bfgs_b to find the MAP estimate of the posterior i.e. the mean of q(w).

bayesian_predictive_distribution finds $\kappa(\sigma_a^2)\mu_a$ i.e. *K_mu* given the data, X, and the parameters of q(w).

The predictive distribution, p, is then found to be expit(*K_mu*) i.e. the logistic function of *K_mu*.

Laplace Approximation

```
import numpy as np
from scipy.special import expit  # logistic fn.

def laplace_parameters(X, var, mean):
    S0_inv = np.linalg.inv( var * np.identity(len(mean)) )
    X_tilde = np.concatenate((np.ones((X.shape[0], 1)), X), 1)
    s = expit(np.matmul(mean, X_tilde.T))
    alpha = np.diag((1-s)*s)
```

```
    scaled_X_tilde = np.matmul(alpha, X_tilde)
    SN_inv = np.matmul( scaled_X_tilde.T, X_tilde) + S0_inv
    SN = np.linalg.inv(SN_inv)

    return mean, SN

def get_b_map(X,y,var):
    X_tilde = np.concatenate((np.ones((X.shape[0], 1)), X), 1)
    b0 = np.ones(X_tilde.shape[1])
    b_map = optimize.fmin_l_bfgs_b(neg_log_posterior, b0, \
        fprime=neg_gradient_log_posterior, args=(X_tilde,y,var))[0]
    return b_map

def bayesian_predictive_distribution(X, SN, mean):
    X_tilde = np.concatenate((np.ones((X.shape[0], 1)), X), 1)
    var_a = np.array([np.matmul(x, np.matmul(SN, x.T)) for x in X_tilde])
    mean_a = np.matmul(mean, X_tilde.T)
    K_mu = mean_a / np.sqrt(1 + np.pi/8 * var_a)

    return K_mu

var_0 = 1
mean = get_b_map(X_RBF_training, y_training, var_0)
mean, SN = laplace_parameters(X_RBF_training, var_0, mean)
p = expit(K_mu)
```

Evidence

The code below approximates the evidence using equation 11.

```
def evidence(X_tilde, y, var):
    b_map = get_b_map(C, y, var)
    M = len(b_map)  # no. hyperparameters
    N = len(y)  # no. data points
    e = np.dot( y, np.log(expit(np.matmul(b_map,X_tilde.T))) )  \
        + np.dot( 1-y, np.log(1 - expit(np.matmul(b_map,X_tilde.T))) )
    e -= 0.5 * M * np.log(N)
    return e
```

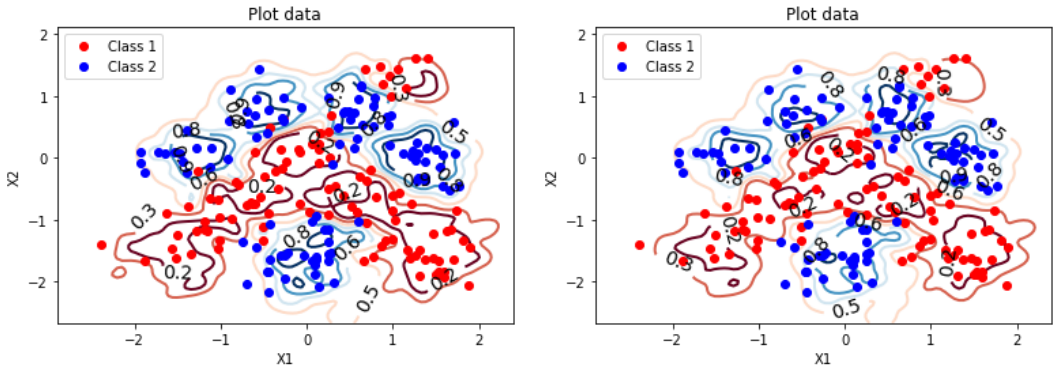
3 Model Performance

3.1 Performance on Base Case

3.1.1 Prediction Contours

The predictive contours for case $l = 0.1$, $\sigma_0^2 = 1$ are shown in figure 1 below, where l is the RBF width and σ_0^2 is the prior variance. Both MAP and full Bayesian solutions are shown. The points on the scatter plot are from the test data while the contours were generated from the training data (hence the contours are predictive). It can be seen that the contours from both MAP and full Bayesian approaches are very similar for

this case. Since the prior is centred on the w_{MAP} this implies that the Gaussian prior is very peaked at its mean like a delta function so that the MAP solution approximates the full Bayesian solution. The contours generated by the two approaches should begin to deviate in similarity when the variance of the prior is increased.



(a) MAP (b) Full Bayesian

Figure 1: Predictive Contour Plots: $l = 0.1$, $\sigma_0^2 = 1$

3.1.2 Log-likelihood estimates and Confusion matrices

Table 1 below compares the average log-likelihood estimates of the MAP case to the full Bayesian case. The MAP cases uses only the w_{MAP} solution to compute the log-likelihood while the full Bayes case uses all values of w with different weightings. w_{MAP} is closer to a likelihood maximising value than all the extreme values that the full Bayesian case takes into account, hence the higher likelihood seen in the MAP case.

	MAP	Bayes
Train	-0.217	-0.256
Test	-0.324	-0.348

Table 1: Average log-likelihood per data point: $l = 0.1$, $\sigma_0^2 = 1$.

MAP

	Prediction	
Truth	0	1
y = 0	0.95	0.05
y = 1	0.05	0.95

Bayes

	Prediction	
Truth	0	1
y = 0	0.95	0.05
y = 1	0.05	0.95

Table 2: Confusion matrices for training data: $l = 0.1$, $\sigma_0^2 = 1$.

	MAP	
	Prediction	
Truth	0	1
y = 0	0.91	0.09
y = 1	0.14	0.86

	Bayes	
	Prediction	
Truth	0	1
y = 0	0.91	0.09
y = 1	0.14	0.86

Table 3: Confusion matrices for test data: l = 0.1, $\sigma_0^2 = 1$.

It can be seen from tables 2 and 3 above that the confusion matrices for the MAP and full Bayesian predictions are the same for the training and test data. This means that, the classification performed equally well when thresholding prediction probabilities at 0.5. This was expected as the MAP and full Bayesian approaches have the same decision boundary as explained at the start of section 2.4.

3.2 Parameter Optimization

The Occam factor implementation produced strange results, with a wide range of log-evidence values, including both large negative values and large positive values. The log-evidence computed using this implementation was found to vary drastically with slight changes in variables l and σ_0^2. Changes of up to 300 were seen by varying σ_0^2 by only 0.4. Hence it was concluded that the implementation was numerically unstable. Consequently, the BIC was used to approximate the evidence. The result of this approximation is shown in figure 2 below.

3.2.1 Grid Search

In the grid search $\ln l$ and $\ln \sigma_0^2$ were varied linearly in the range -2 to 2. The logs were varied as the model's dependency on both of these parameters is exponential.

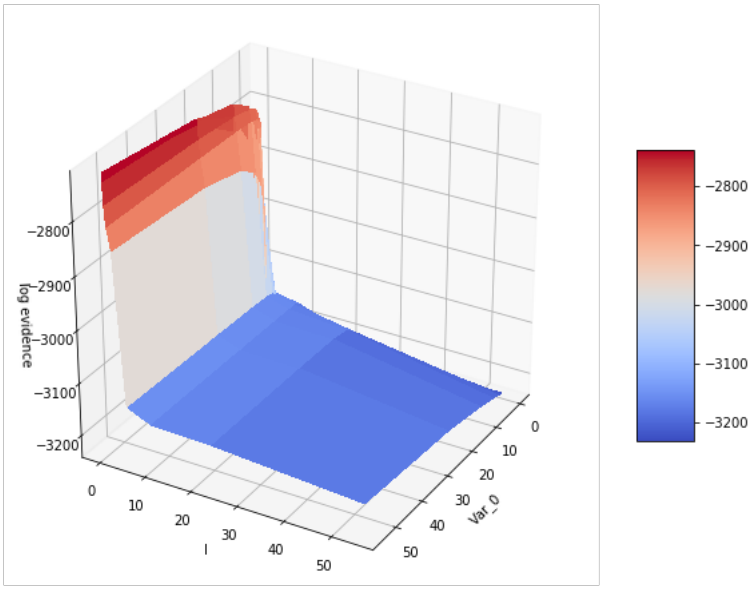
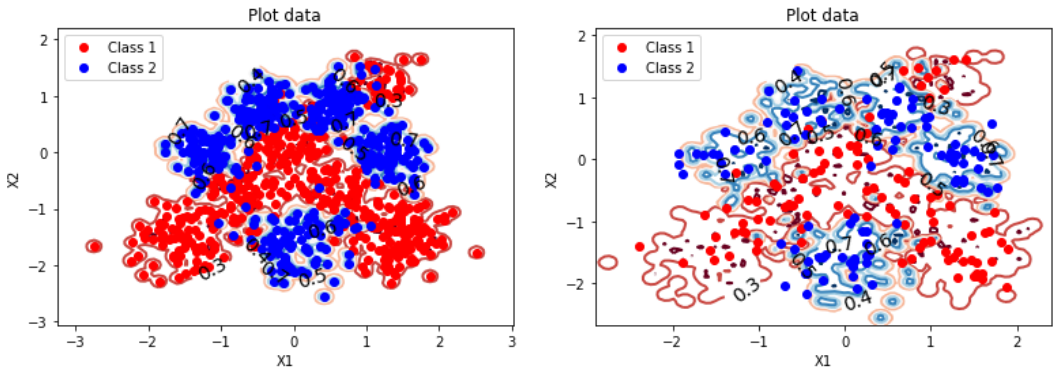


Figure 2: Log evidence vs. l and σ_0^2

It can be seen that the evidence increases with increasing σ_0^2 and decreasing l. This increase in evidence with σ_0^2 is due to over-fitting of the model to training data and therefore this grid search cannot be used to reliably pick an optimum value of parameters σ_0^2 and l. Low l maximises the likelihood as this essentially makes the radial basis function appear as deltas on the data points. High σ_0^2 maximises the evidence as this makes the prior tend to a uniform distribution so that it has no effect on the posterior and that the likelihood provides the only contribution to the posterior. Evidence of this over-fitting can be seen in the contour plots below taking values l = 0.05 and σ_0^2 = 100.



(a) Training (b) Test

Figure 3: Predictive Contour Plots: l = 0.1, $\sigma_0^2 = 1$

It can be seen that the contours closely match the training data in fig 3(a). The contours perfectly outline the edges of the data points, leaving very little white space. In fig. 3(b) the test data was plotted against the contours generated by the training data, so as to visually evaluate the predictive ability of the full Bayesian approach using

these parameters. It can be seen that some of the blue dots lie in the red region and vice-versa. Hence the predictions made are poor. This is evidence of the over-fitting of the model to the training data.

4 Conclusions

- For the base case, the MAP and full Bayesian predictive approaches produced very similar probability contours as the Gaussian prior was very peaked.
- Occam factor computation was numerically unstable so could not be used to compute evidence.
- BIC-based evidence optimisation via grid search was unreliable as it over-fit the training data.
- The MAP and full Bayesian predictive methods classified the test data equally well.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Appendix: Short Lab Report

1. Introduction

The data set provided consisted of a list of data points, $\mathbf{x^{(n)}}$, and a classification, $y^{(n)}$, for each point where n denoted the index of the point and N the total number of data points given. Each point was a row vector, so a column vector of row vectors formed a matrix, X. A list of classifications formed a row vector $\mathbf{y}$. In the data given each data point had two components while each classification was either a 1 or 0. The main task was to predict if each point from a test data set belonged to class 1 ($y^{(n)} = 0$) or class 2 ($y^{(n)} = 1$) given that the classifier had been trained on different data set. The first part of the lab involved the use of a logistic classifier which created a single linear boundary partitioning the data into the two classes. The second part of the lab made use of a set of radial basis functions (RBF) to create what was effectively multiple non-linear class boundaries. In both parts of the lab contours were plotted, giving the probability that data points on it belonged to class 2. In this report the class boundary corresponds to the 0.5 probability contour, as a 0.5 threshold was set when the confusion matrices were calculated.

2. Logistic classification Theory

A set of parameters β was defined. This was used to form a linear combination of the features of $\mathbf{x^{(n)}}$. β may be thought of as a vector in feature space so by taking the dot product with each data point a measure of the perpendicular distance from the decision boundary was obtained. The resulting dot product could have taken any value on the number line. The logistic function mapped this infinite interval to the interval [0,1]. This was therefore used to specify the probability that a data point belonged to class 2. How was β picked though? Given that it was known that $p(y^{(n)} = 1|x^{(n)}) = \sigma(\beta^T \mathbf{x^{(n)}})$ and that $p(y^{(n)} = 0|x^{(n)}) = 1 - \sigma(\beta^T \mathbf{x^{(n)}})$ the likelihood was defined compactly as given on the first page of the lab handout. Gradient ascent on β was done to maximize the likelihood. The explanation for this is given in the 'Gradient Ascent' section.

3. Radial Basis Function (RBF) Expansion Theory

The logistic classifier did a very poor job in classifying the data, as can be seen in the probability contour plots (see section 'Plots') and the confusion matrix (table 1). Gaussian basis functions were used as an improvement. They were referred to as radial basis functions as the output depended only on the distance to a given centre. In this lab, each data point was expanded by radial basis functions centred on all the data points. The $\mathbf{(m + 1)}$th feature of a given data point x^n was given by a Gaussian centred on the mth data point. This meant that if the $\mathbf{m}$th data point was far far from the nth data point, then the $\mathbf{(m + 1)}$th feature was small, but it was large if the point was nearby. The variance of the Gaussians were related to a parameter l, that was varied in the activities. These expanded data points were then acted on by the parameters β, so that a linear combination of a set of non-linear Gaussian functions was taken. These were then fed into the logistic classifier just as in the first part of the lab. The effect was that the contours were linear in the expanded feature space i.e.

they formed hyper planes in this expanded feature space. These mapped to non-linear contours in the original feature space. Theoretically these new probability contours may have been found by setting the output of the logistic function to a constant. Since the logistic function performs one-to-one mapping this required the inputs to be constant too. The input is a linear combination of Gaussian functions. Therefore the contouring algorithm assigned the same probability to points that had equal result from this linear combination of Gaussians. What did this mean? It meant that the contours had a tendency to encircle clusters of data of a given class. As long as one data point was close to many neighbours, its linear combination produced a similarly high value. Points far away from any cluster were given low probability. Consequently the radial basis functions had a tendency to encircle clusters of data very nicely. This had the effect of improving the classification process as can be seen in figure 2 in the 'Plots' section.

4. Gradient ascent

This algorithm found the value of β that maximized the likelihood using the following expression:

$$\beta_{new} = \beta_{old} + \eta \frac{\partial L}{\partial \beta} \tag{13}$$

where η was the learning rate and L, the log likelihood. Differentiating L gave the expression:

$$\beta_{new} = \beta_{old} + \eta(y - \sigma(\beta_{old} X^T))X \tag{14}$$

For easy translation to Python code, all vectors in the above equation were given in row form and the matrix X was given as a vector of row vectors. The derivation of $\frac{\partial L}{\partial \beta}$ is given in appendix A and the code for the gradient ascent are given in *estimate_parameters* in appendix B.

The learning rate was set by trial-and-error to find one that converged on the optimum β in the shortest time. This was done by increasing it until oscillations appeared in the progression of the likelihood. Then the maximum learning rate which did not produce oscillations was used. With this selected learning rate, the number of iterations required to converge to the function's maximum was then set.

5. Logistic Classification Results and Discussion

The average log-likelihood per point was plotted against iteration steps as the optimization of beta proceeded. The result of this optimization is given in figure 1 in the 'Plots' section for both the training and test data sets. It can be seen that the likelihood was maximized at a values of -0.61 and -0.66 for the training and test data sets respectively.

The optimum parameters obtained from the training data were then used to make predictions on the test data, generating a confusion matrix. The code for this is given

in *generate_confusion* in appendix B. The confusion matrix gives the proportions of true and false positives and negatives compared to the size of the test data set. The resulting confusion matrix and the ideal confusion matrix are given in tables 1 and 2 below.

		Predicted label, ŷ	
		0	1
Label, y	0	true negatives: 0.35	false positives: 0.155
	1	false negatives: 0.18	true positives: 0.315

Table 2: Confusion matrix: logistic classifier

		Predicted label, ŷ	
		0	1
Label, y	0	true negatives: 0.505	false positives: 0.0
	1	false negatives: 0.0	true positives: 0.495

Table 3: Ideal confusion matrix for dataset

It can be seen in the 'Plots' section that the logistic classifier performed very poorly as the classes were intermixed making a linear decision boundary very inappropriate. The high false rates (16% and 18%) seen in the confusion matrix therefore come as no surprise.

6. Radial Basis Expansion Results and Discussion

Again, the average log-likelihood per data point was plotted as optimization proceeded. The final training and test log-likelihood per data point for each of the three cases (l = 0.01, 0.1, 1) are given in table 3. The confusion matrices obtained are given in tables 4 to 6.

	training	test
l=0.01	-0.00714	-0.6553
l=0.1	-0.128	-0.284
l=1	-0.208	-0.226

Table 4: Maximum log-likelihoods for RBF expanded cases

For l=0.01 it can be seen that the log-likelihood for the training data is very close to 0. The model has been over-fit to the training data resulting in the poor log-likelihood seen in the test data predictions. For l=-0.1 it can be seen that the training data is still slightly over-fit as there is a considerable difference (0.156) between its log likelihood and that of the test data. The l=1 case shows no signs of over-fitting since the maximum test and training log-likelihood values are the similar. This case also gives the largest log-likelihood on the test data out of the three cases. The l=1 case therefore performs the best in making predictions on the test data.

		Predicted label, ŷ	
		0	1
Label, y	0	true negatives: 0.49	false positives: 0.015
	1	false negatives: 0.445	true positives: 0.005

Table 5: Confusion matrix: l=0.01

		Predicted label, ŷ	
		0	1
Label, y	0	true negatives: 0.445	false positives: 0.06
	1	false negatives: 0.055	true positives: 0.44

Table 6: Confusion matrix: l=0.1

		Predicted label, ŷ	
		0	1
Label, y	0	true negatives: 0.445	false positives: 0.05
	1	false negatives: 0.04	true positives: 0.445

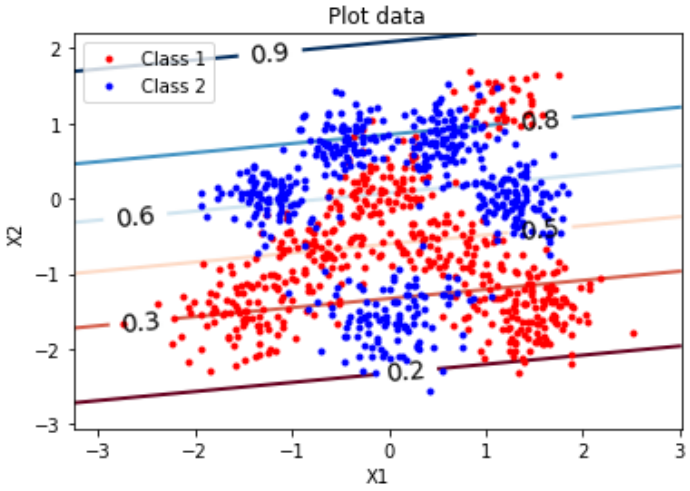
Table 7: Confusion matrix: l=1

It can be seen that the l=0.01 case performs very poorly as the variance of the Gaussian basis functions is too small. This gives probability contours that are centred on each point in the training data set, resulting in very poor predictions on the test data set. Hence, this case performed much worse than the logistic classifier with false rates of 2% and 45%.

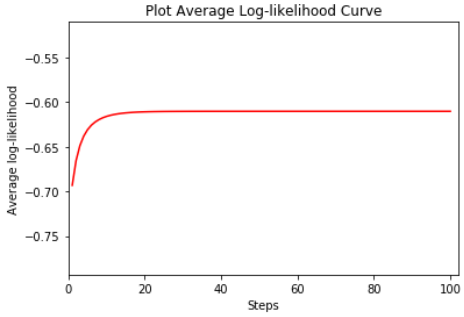
The l=0.1 and l=1 cases performed much better than the previous two cases as in these cases the Gaussian mass is not concentrated on data points. From l=0.1 to l=1 very little reduction is seen in false rates so we have diminishing returns after increasing l past 0.1.

7. Plots

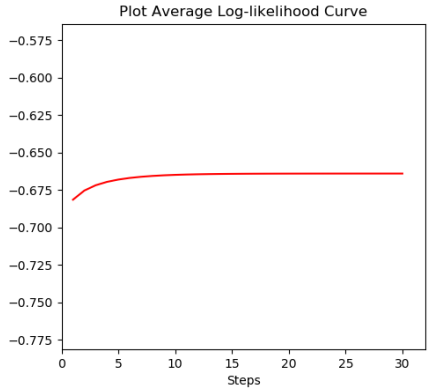
All predictions made were visualized as probability contours on the data set. Figure ... below shows that the contours for the logistic classifier are linear (as mentioned in the 'Logistic Classification Theory' section). This was clearly not ideal for this data set. It can be seen that RBF expansion produced much better contours for the cases l=0.1 and l=1.



(a) Probability contours for logistic classifier

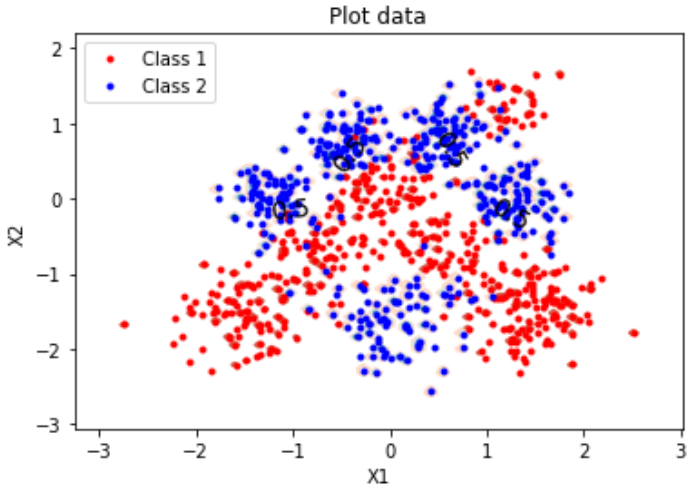


(b) Training set log-likelihood vs. # iterations

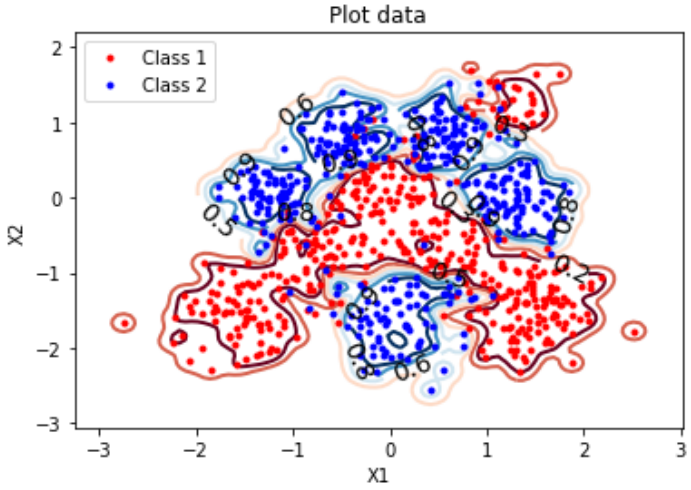


(c) Test set log-likelihood vs. # iterations

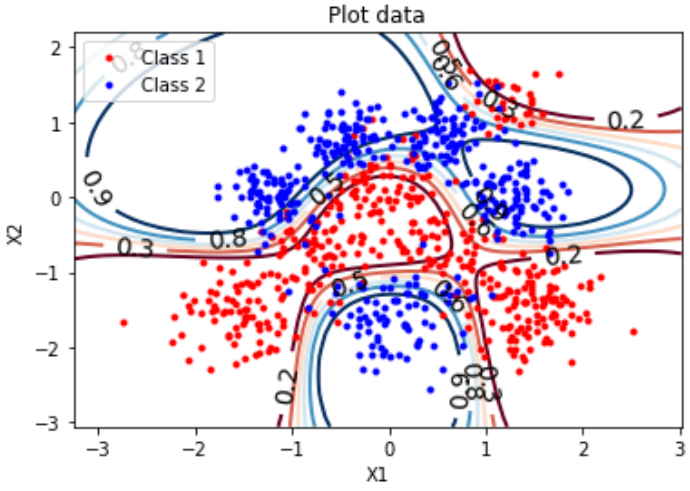
Figure 1: Plots for non-expanded data



(a) Probability contours: $l = 0.01$, $\eta = 0.01$



(b) Probability contours: $l = 0.1$, $\eta = 0.01$



(c) Probability contours: $l = 1$, $\eta = 0.0001$

Figure 2: Contours for RBF expanded data

A: Derivations

Defining all vectors as column vectors, so that X is a row vector of column vectors, $\mathbf{x^{(n)}}$:

$$L(\beta) = log(p(y|X,\beta) = \prod_{n=1}^{N} \sigma(\beta^T \mathbf{x^{(n)}})^{y^{(n)}} (1 - \sigma(\beta^T \mathbf{x^{(n)}}))^{1-y^{(n)}}$$

$$= \sum_{n=1}^{N} log\left(\sigma(\beta^T \mathbf{x^{(n)}})^{y^{(n)}} (1 - \sigma(\beta^T \mathbf{x^{(n)}}))^{1-y^{(n)}}\right)$$

$$= \sum_{n=1}^{N} y^{(n)} log\left(\sigma(\beta^T \mathbf{x^{(n)}})\right) + (1 - y^{(n)}) log\left((1 - \sigma(\beta^T \mathbf{x^{(n)}}))\right)$$

$$\frac{\partial L}{\partial \beta} = \sum_{n=1}^{N} y^{(n)} (1 - \sigma(\beta^T \mathbf{x^{(n)}})) \mathbf{x^{(n)}} - (1 - y^{(n)}) (\sigma(\beta^T \mathbf{x^{(n)}})) \mathbf{x^{(n)}}$$

$$\frac{\partial L}{\partial \beta} = \sum_{n=1}^{N} \left(y^{(n)} - \sigma(\beta^T \mathbf{x^{(n)}})\right) \mathbf{x^{(n)}}$$

$$\frac{\partial L}{\partial \beta} = \sum_{n=1}^{N} \left(y^{(n)} - \sigma(\beta^T \mathbf{x^{(n)}})\right) \mathbf{x^{(n)}}$$

$$\frac{\partial L}{\partial \beta} = X(y - \sigma(\beta^T X))$$

Then for easy translation to Python code all vectors may be written in column form, so that X is a column vector of row vectors. This gives:

$$\frac{\partial L}{\partial \beta} = (y - \sigma(\beta X^T))X$$

B: Program Listings

```
def estimate_parameters(X, y):
    X = np.concatenate((np.ones((X.shape[ 0 ], 1 )), X), 1)
    a = 1e-3  # learning rate
    b = np.zeros(X.shape[1])  # initialise b
    P = 100  # no. iterations
    ll = np.zeros(P)

    for i in range(P):
        b += a * np.dot((y - logistic(np.dot(b,X.T))), X)
        ll[i] = compute_average_ll(X, y, b)
```

```
    return ll, b

def get_confusion(X, w):
    X = np.concatenate((np.ones((X.shape[0], 1)), X), 1)
    label = [1 if item > 0.5 else 0 for item in logistic(np.dot(X, w))]

    confusion = np.array([[0, 0], [0, 0]])

    for i in range(len(y_test)):
        if y_test[i] == 0:
            if label[i] == 0:
                confusion[0, 0] += 1
            else:
                confusion[0, 1] += 1
        else:
            if label[i] == 0:
                confusion[1, 0] += 1
            else:
                confusion[1, 1] += 1

    confusion = confusion / len(y_test)

    return confusion
```