CUED - Engineering Tripos Part IIA 2017-2018 **Module Laboratory Report**

Student name:		CRSID:		Module number:	

Feedback to the student ☐ **See also comments in the text**		Very good	**Good**	Needs improvmt
C O N T E N T	**Completeness, quantity of content:** Has the report covered all aspects of the lab? Has the analysis been carried out thoroughly?			
	Correctness, quality of content Is the data correct? Is the analysis of the data correct? Are the conclusions correct?			
	Depth of understanding, quality of discussion Does the report show a good technical understanding? Have all the relevant conclusions been drawn?			
	Comments:			
P R E S E N T A T I O N	**Attention to detail, typesetting and typographical errors** Is the report free of typographical errors? Are the figures/tables/references presented professionally?			
	Comments:			

Raw report mark	**/ 5**
Penalty for lateness	

The weighting of comments is not intended to be equal, and the relative importance of criteria may vary between modules. A good report should attract 4 marks.

1 mark / week or part week.
Please use allowance forms to inform the teaching office about mitigating circumstances.

Marker: Date:

ENGINEERING TRIPOS PART II A

3F8 Inference Coursework
Short Report

Name: Gregory Brooks

College: Christ's
Source code at `https://github.com/Gregox273/3f8_lab`

1. **Preparation Exercises**

   (a) *Derive the gradients of the log-likelihood of the parameters:*

$$\begin{aligned}
\frac{\partial \mathcal{L}(\beta)}{\partial \beta} &= \frac{\partial}{\partial \beta} log(p(y|X,\beta)),\ \textit{where} \\
p(y|X,\beta) &= \prod_{n=1}^{N} \sigma(\beta^T \tilde{x}^{(n)})^{y^{(n)}} (1-\sigma(\beta^T \tilde{x}^{(n)}))^{1-y^{(n)}} \\
&= \prod_{n=1}^{N} (1+exp(-\beta^T \tilde{x}^{(n)}))^{-y^{(n)}} \left(1-\frac{1}{1+exp(-\beta^T \tilde{x}^{(n)})}\right)^{1-y^{(n)}} \\
\therefore log(p(y|X,\beta)) &= \sum_{n=1}^{N} -y^{(n)} log(1+exp(-\beta^T \tilde{x}^{(n)})) + (1-y^{(n)}) log\left(1-\frac{1}{1+exp(-\beta^T \tilde{x}^{(n)})}\right) \\
&= \sum_{n=1}^{N} -y^{(n)} log(1+exp(-\beta^T \tilde{x}^{(n)})) + (1-y^{(n)}) log\left(\frac{exp(-\beta^T \tilde{x}^{(n)})}{1+exp(-\beta^T \tilde{x}^{(n)})}\right) \\
&= \sum_{n=1}^{N} -log(1+exp(-\beta^T \tilde{x}^{(n)})) + (1-y^{(n)})(-\beta^T \tilde{x}^{(n)}) \\
&= \sum_{n=1}^{N} -log(1+exp(-\beta^T \tilde{x}^{(n)})) - log(exp(\beta^T \tilde{x}^{(n)})) + y^{(n)}(\beta^T \tilde{x}^{(n)}) \\
&= \sum_{n=1}^{N} -log(1+exp(\beta^T \tilde{x}^{(n)})) + y^{(n)}(\beta^T \tilde{x}^{(n)}) \\
&= \sum_{n=1}^{N} -log(1+exp\left(\beta_0 + \sum_{d=1}^{D} \beta_d x_d^{(n)}\right)) + y^{(n)}\left(\beta_0 + \sum_{d=1}^{D} \beta_d x_d^{(n)}\right) \\
\therefore \frac{\partial \mathcal{L}(\beta)}{\partial \beta_0} &= \sum_{n=1}^{N} y^{(n)} - \frac{exp\left(\beta_0 + \sum_{d=1}^{D} \beta_d x_d^{(n)}\right)}{1+exp\left(\beta_0 + \sum_{d=1}^{D} \beta_d x_d^{(n)}\right)} = \sum_{n=1}^{N} y^{(n)} - \sigma(\beta^T \tilde{x}^{(n)}) \\
\frac{\partial \mathcal{L}(\beta)}{\partial \beta_{i\neq 0}} &= \sum_{n=1}^{N} x_i^{(n)} y^{(n)} - \frac{x_i^{(n)} exp\left(\beta_0 + \sum_{d=1}^{D} \beta_d x_d^{(n)}\right)}{1+exp\left(\beta_0 + \sum_{d=1}^{D} \beta_d x_d^{(n)}\right)} = \sum_{n=1}^{N} x_i^{(n)}(y^{(n)} - \sigma(\beta^T \tilde{x}^{(n)}))
\end{aligned}$$

(b) *Pseudocode implementation of gradient ascent using* $\beta^{(new)} = \beta^{(old)} + \eta \frac{\partial}{\partial \beta}\mathcal{L}(\beta^{(old)})$

```
def dL_db(b,y,X):
    a = exp(X dot b)
    b = a / (1+a)    # = sigma(X dot b)
    c = y − b
    return (c dot X)

for (number of iterations):
    b += rate ∗ dL_db(b,y_tr,x_tr)
return b
```

Listing 1: Pseudocode gradient ascent implementation

The learning rate η must be chosen to be small enough that the algorithm converges on a solution without overshooting and/or diverging, but large enough that a good approximation for the solution can be reached within a reasonable number of iterations (to reduce the time taken for the algorithm to run).

2. **Remaining Exercises**

(c) *Visualisation of Supplied Data*

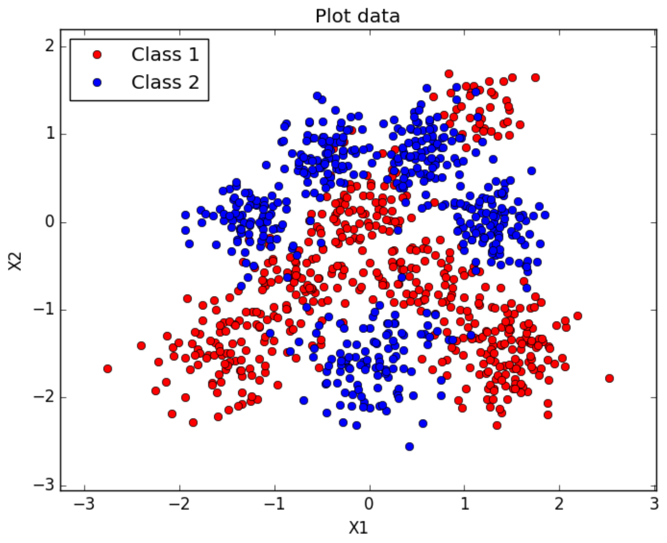


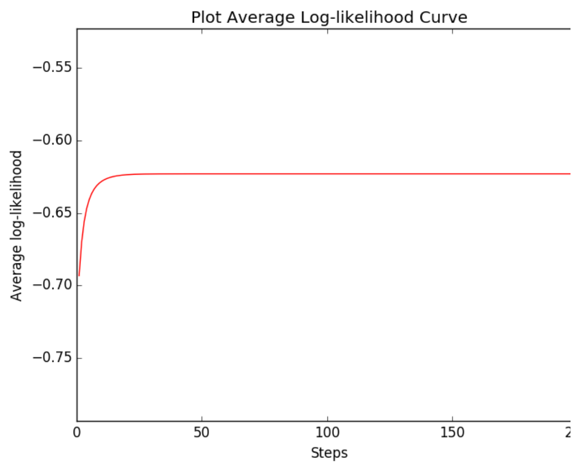
Figure 1: Plot of supplied data

A classifier with a linear boundary is unlikely to perform well here as the data points for the two classes appear fairly mixed together when plotted as in Figure 1, with no way of separating the two class regions using a single straight line through the input space.

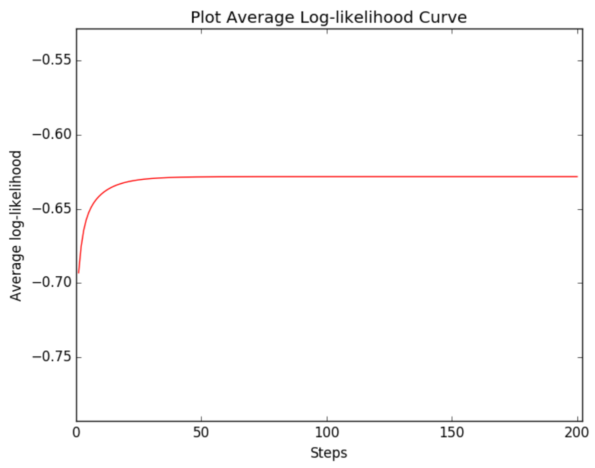
(d) *Splitting Into Training and Test Data*
The supplied data was split in a 70:30 ratio by randomly selecting 30% of the data points to be used as test data (and the rest for training). In numpy this was achieved by creating a random (shuffled) boolean mask vector to mark 300 out of the 1000 data points to be reserved

for testing. A 70:30 ratio was chosen as an acceptable tradeoff between classifier accuracy (which improves with the size of the training dataset, as the classifier must be trained on data which is representative of all the inputs it is likely to come across in use) and classifier error estimate (which improves in accuracy with a larger test dataset). In practice, the test dataset does not have to be as large as the training dataset to reach an acceptable estimate of classifier accuracy.

(e) *Training of Logistic Classification Method*



(a) Training data



(b) Test data

Figure 2: Training curves showing average log likelihood per datapoint

Listing 2: Python implementation of gradient ascent estimation of β

```
def dL_db(b,y,X):
        a = np.exp(np.dot(X,b))
        b = a / (1 + a)
        c = y - b
        rtn = np.dot(c,X)
        return rtn

def train(iterations, rate, ll_train, ll_test, x_tr, y_tr, x_t, y_t,
    b):
        for count in range(0,iterations):
                # Calculate ll
                ll_train.append( compute_average_ll(x_tr, y_tr, b) )
                ll_test.append( compute_average_ll(x_t, y_t, b) )
                # Update beta
                b += rate * dL_db(b,y_tr,x_tr)
        return b

beta = train(iterations, l_rate, ll_train, ll_test, XX_train,
   y_train, XX_test, y_test, beta)  # Example usage
```

(f) *Classifier Performance*

	Training	**Test**
Final log-likelihood	-0.623039	-0.628198

Table 1: Final average log likelihood per data point

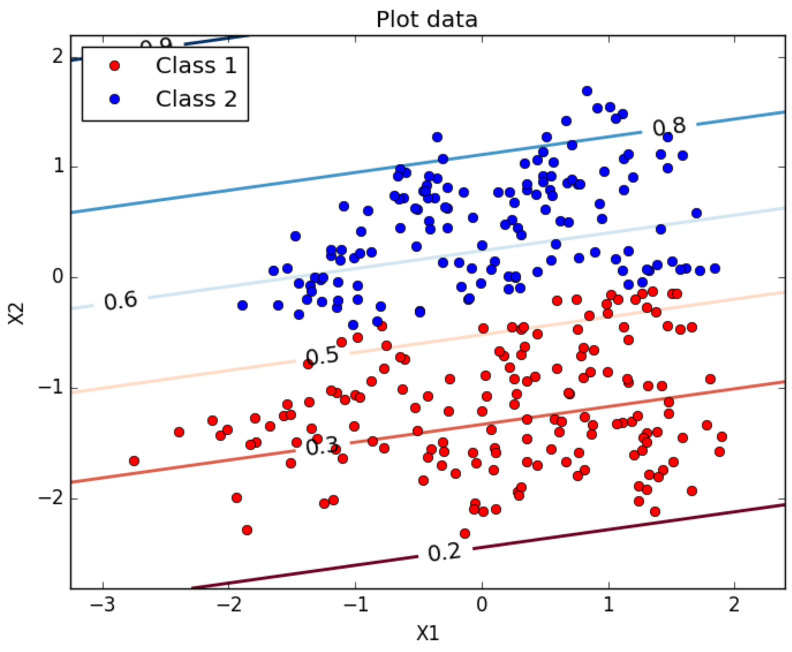


Figure 3: Result of applying classifier to test data

The confusion matrix was calculated to be:

$$\begin{bmatrix} 0.76642336 & 0.23357664 \\ 0.33128834 & 0.66871166 \end{bmatrix}$$

(g) *Expanded Inputs*

RBF width l	**Learning rate η**
0.01	0.005
0.1	0.0002
1	0.0002

Table 2: Table of RBF widths investigated and corresponding empirically determined learning rates (for 9999 iterations of gradient ascent)

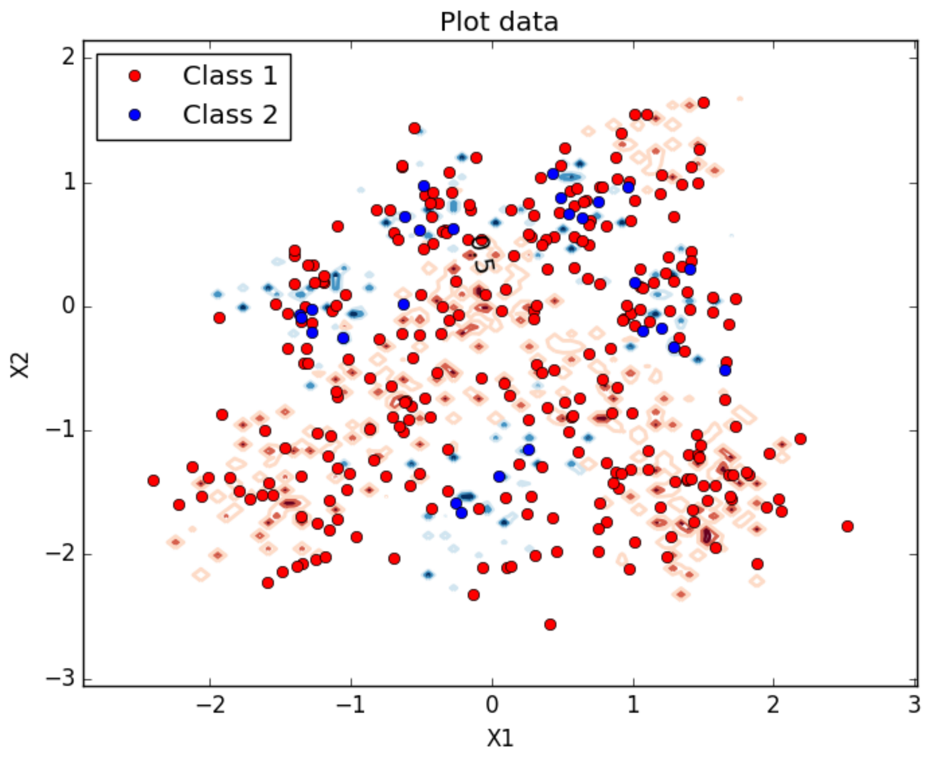


Figure 4: Results of applying classifier to test data with RBF width l = 0.01

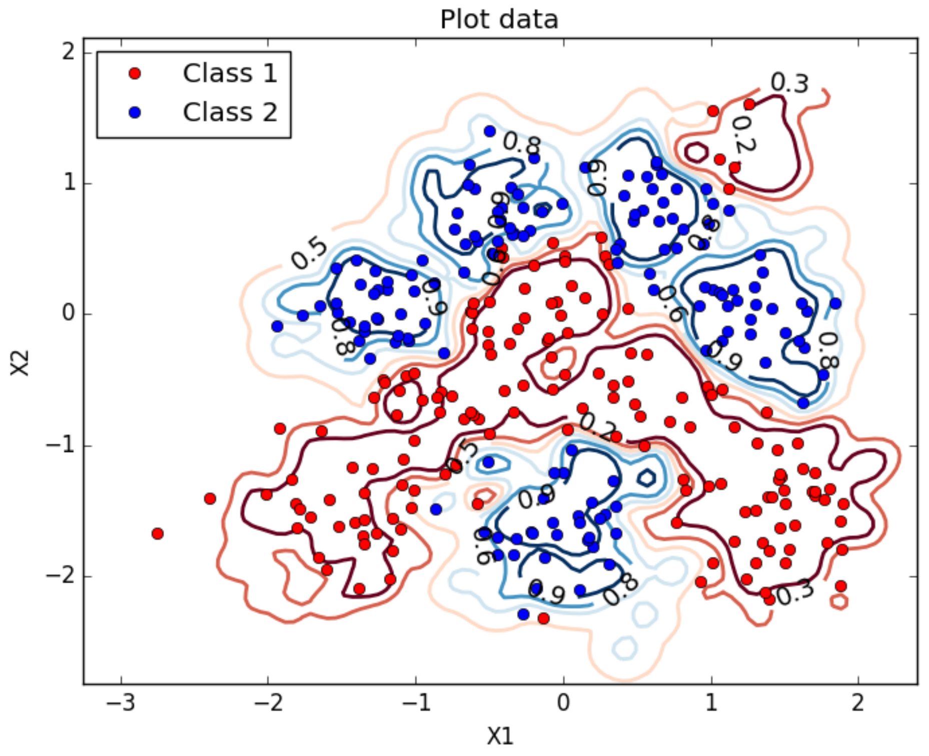


Figure 5: Results of applying classifier to test data with RBF width l = 0.1

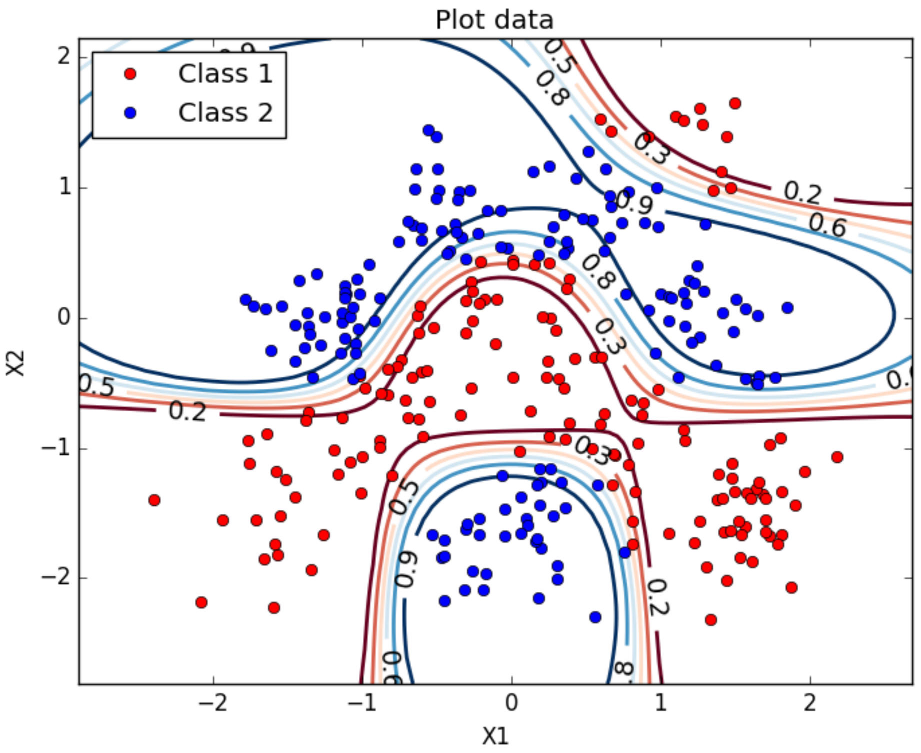


Figure 6: Results of applying classifier to test data with RBF width l = 1

(h) **Expanded Inputs Continued**

l	Training LL	Test LL	Confusion Matrix
0.01	-0.021443	-0.664425	$\begin{bmatrix} 0.97297297 & 0.02702703 \\ 0.85526316 & 0.14473684 \end{bmatrix}$
0.1	-0.156120	-0.246934	$\begin{bmatrix} 0.95302013 & 0.04697987 \\ 0.09933775 & 0.90066225 \end{bmatrix}$
1	-0.208938	-0.191595	$\begin{bmatrix} 0.93243243 & 0.06756757 \\ 0.05921053 & 0.94078947 \end{bmatrix}$

Table 3: Final average log likelihood per data point for a range of RBF widths l

The classifier performs much more accurately using expanded inputs, with log likelihoods closer to zero and a far smaller proportion of false results in the confusion matrices (for l > 0.01 at least), compared to the linear classifier. This makes sense as the distribution of the input data did not allow for a linear class boundary to separate the two classes, whilst expanding the inputs allows the nonlinear class boundaries shown in Figures 4 to 6 to be defined as regions close to the training data points (with closeness defined by the RBF functions).