ENGINEERING TRIPOS PART IIA

3F8 LOGISTIC CLASSIFICATION LAB REPORT

Matthew Coates (mc955) **Pembroke College** **Lab Date: 16th Feb 2018**

1 Logistic Classification

The objective of this experiment was to implement a classifier, apply it to a simple dataset, and evaluate the results using several metrics. The classifier was to be trained using the batch gradient ascent rule applied to the model parameters. In order to apply the gradient ascent method the gradient of the log-likelihood of the parameters must first be found.

1.1 Derivation of the gradient of $\mathcal{L}(\beta)$

In this model each data point's class label is assumed to be generated from a Bernoulli distribution in which the probability of a positive class label is given by a logistic function applied to weighted inputs. This gives the following distribution.

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

The log-likelihood of the parameters is then just $\mathcal{L}(\beta) = \log(P(y|X,\beta))$.

$$\mathcal{L}(\beta) = \sum_{n=1}^{N} y^{(n)}\log(\sigma(\beta^T\tilde{x}^{(n)})) + (1-y^{(n)})\log(1-\sigma(\beta^T\tilde{x}^{(n)})) \tag{2}$$

In order to compute the gradient of $\mathcal{L}(\beta)$ the following identities must be used.

$$\frac{\partial}{\partial\beta}\big(\log(\sigma(\beta^T\tilde{x}^{(n)}))\big) = \tilde{x}^{(n)}(1-\sigma(\beta^T\tilde{x}^{(n)})) \tag{3}$$

$$\frac{\partial}{\partial\beta}\big(\log(1-\sigma(\beta^T\tilde{x}^{(n)}))\big) = -\sigma(\beta^T\tilde{x}^{(n)})\tilde{x}^{(n)} \tag{4}$$

This yields the following expression for the gradient of $\mathcal{L}(\beta)$:

$$\frac{\partial\mathcal{L}(\beta)}{\partial\beta} = \sum_{n=1}^{N} y^{(n)}(1-\sigma(\beta^T\tilde{x}^{(n)}))\tilde{x}^{(n)} - (1-y^{(n)})\sigma(\beta^T\tilde{x}^{(n)})\tilde{x}^{(n)} \tag{5}$$

$$= \sum_{n=1}^{N} \big(y^{(n)} - \sigma(\beta^T\tilde{x}^{(n)})\big)\tilde{x}^{(n)} \tag{6}$$

1.2 Gradient Ascent

The batch gradient ascent method was implemented to train the classifier. This method tries to maximise the log-likelihood of the weights, $\mathcal{L}(\beta)$. The weights, β, are initialised to one and then subsequently updated according to the following rule.

$$\beta^{\text{new}} = \beta^{\text{old}} + \eta \frac{\partial \mathcal{L}(\beta)}{\partial \beta} \tag{7}$$

Here $\eta > 0$ is the learning rate. In order to implement gradient ascent efficiently, equation 6 above can be vectorised which gives the following update rule for the classifier weights.

$$\beta^{\text{new}} = \beta^{\text{old}} + \eta\, X^T\big(\mathbf{y} - \sigma(X.\beta)\big) \tag{8}$$

The python code implementation of the training algorithm is given below.

```
# Gradient Ascent
for i in range (0, training_steps):

    # Compute dL(w)/dw
    dw = np.dot(np.transpose(X),y-logistic(np.dot(X,w)))

    # Update Weights
    w = w + eta*dw
```

1.3 Dataset Visualisation

The data is made up of X, a 1000 × 2 dimensional array containing two-dimensional input features, and y, a 1000 dimensional binary vector containing the class labels. This is the dataset that the classifier will be applied to. First the dataset was visualised by plotting it in the two-dimensional input space whilst displaying each data point's class label.

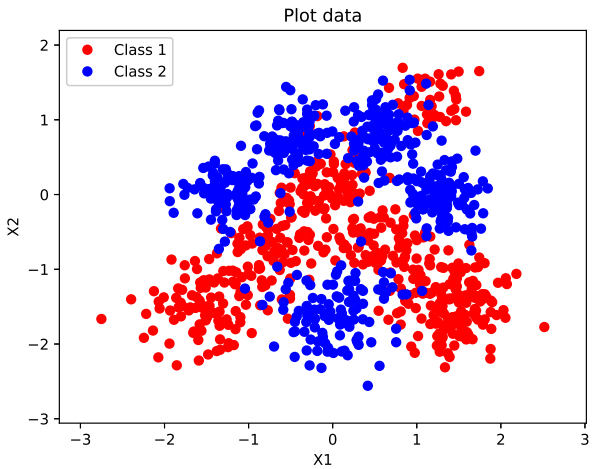


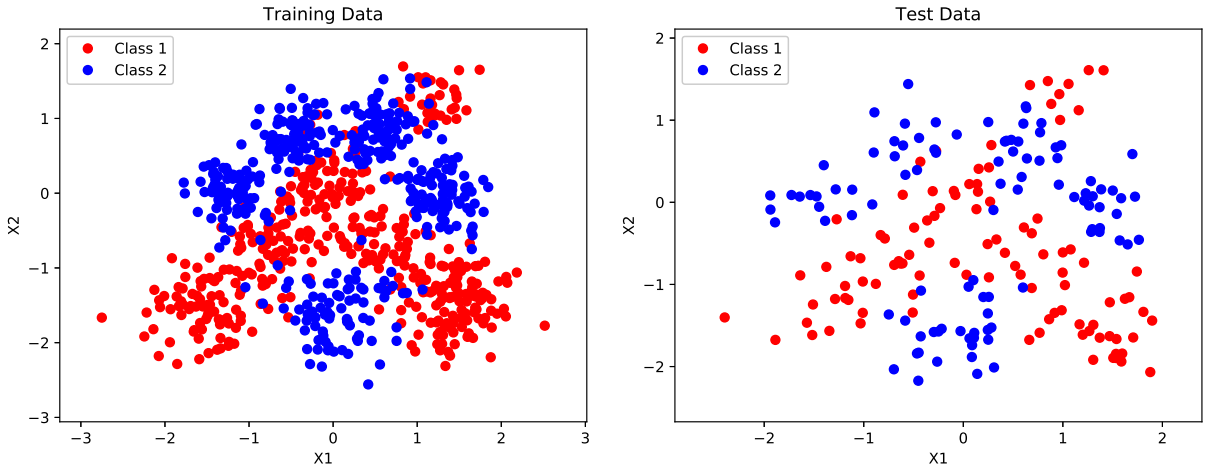
Figure 1: Dataset Visualisation

It is clear from figure 1 that the data points would be classified poorly using a linear class boundary. By observation, a linear class boundary would at best be able to classify roughly 70% of data points correctly, which is not desirable.

1.4 Training Data Selection

In order to be able to evaluate the performance of the classifier on unseen data it was required to split the dataset into training data and test data. Training data is used for learning, which in this case is fitting the parameters of the classifier. The test data is a dataset that is independent of the training dataset, but that follows the same probability distribution as it. If the classifier fits the test dataset well, minimal overfitting has taken place. A classifier that performs better on the training dataset as opposed to the test dataset has usually undergone overfitting. The test dataset therefore is able to asses the performance and generalisation of the fully specified classifier.

In order to train the classifier well a large number of training data points are required. However to evaluate the classifier a representative sample of the overall dataset is required as test data. For these reasons the training data was taken to be the first 800 entries of the dataset, and the test data the remaining 200 entries. These datasets have been visualised below.



(a) Training Data

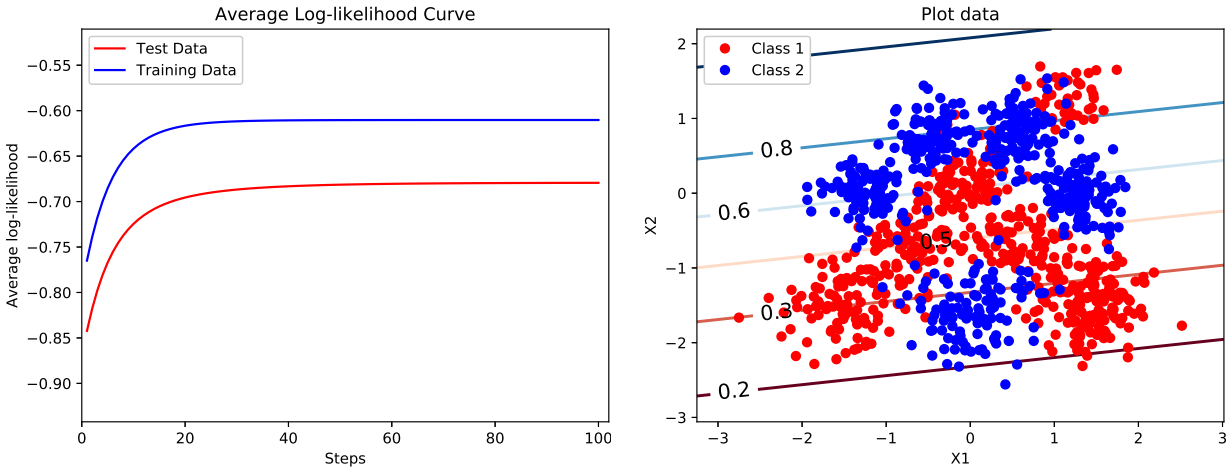
(b) Test Data

Figure 2: Classification Datasets

From figure 2 it can be seen that the test data contains a representative sample of the overall dataset. Therefore our selection of test and training data from the original dataset is acceptable and we can proceed with training the classifier.

1.5 Classifier Training

The gradient ascent method outlined in section 1.2 was used to train the classifier on the training dataset. The training curve showing the averaged log-likelihood for the training and test datasets as the optimisation proceeds is shown below. The predictions of the classifier were then visualised by overlaying probability contours onto the plot of figure 1.



(a) Training Curve

(b) Prediction Visualisation

Figure 3: Logistic Classifier Output

1.6 Classifier Performance

The performance of the above classifier was then characterised. The final log-likelihood values, averaged over all data points, for the training and test datasets were found to be $\mathbf{-0.61}$ and $\mathbf{-0.68}$ respectively. The confusion matrices for each of the datasets were also found.

$$\text{Training Data} = \begin{bmatrix} 0.74 & 0.26 \\ 0.28 & 0.72 \end{bmatrix} \qquad \text{Test Data} = \begin{bmatrix} 0.66 & 0.34 \\ 0.33 & 0.67 \end{bmatrix}$$

Because the model is trained on the training data, there will always be a decrease in performance when the classifier is applied to the test data. Here a linear classification boundary was used, which as we hypothesised in section 1.3, has been unable to classify the dataset well. In the case of the test data, 66% of the data points were classified correctly, a decrease from 73% observed with the training data. The fact that this decrease in performance is only 7% suggests that minimal overfitting of the parameters has occurred.

2 Radial Basis Function Expansion

In order to obtain a non-linear classification boundary the input data was expanded through a set of radial basis functions centred on the training datapoints.

2.1 Classifier Training and Performance

The radial basis function (RBF) was controlled by a hyper-parameter l which varied its width. The logistic classification model was then trained on the feature-expanded inputs as in section 1.5 for different values of l. The learning rate η was adjusted appropriately for each l. The resulting predictions have been visualised below.

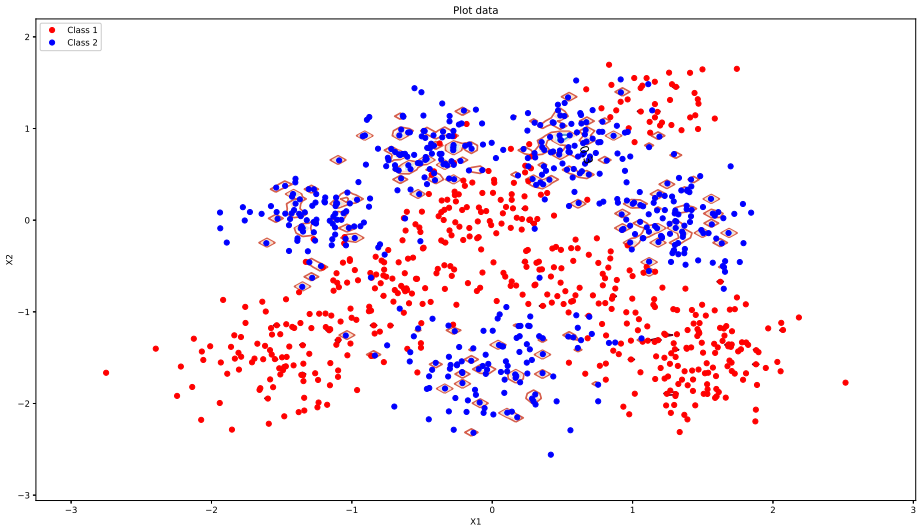


Figure 4: Prediction Visualisation for $l = 0.01$

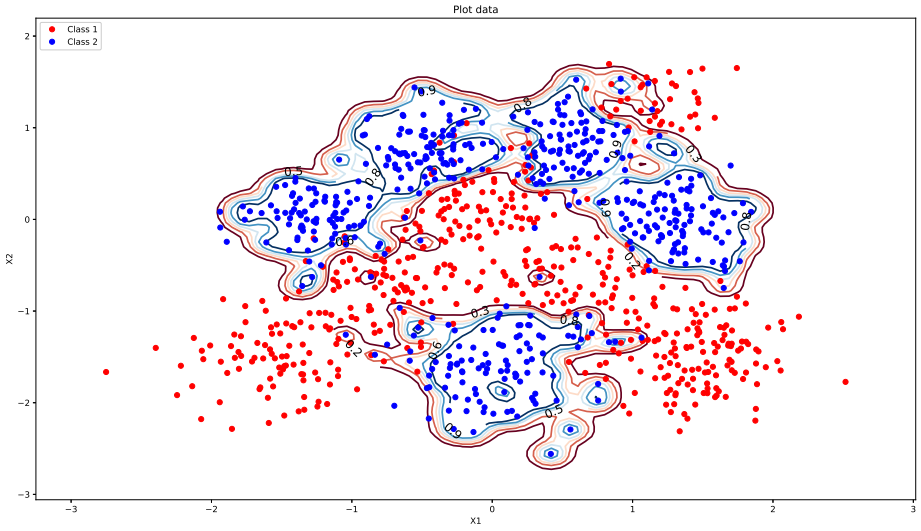


Figure 5: Prediction Visualisation for $l = 0.1$

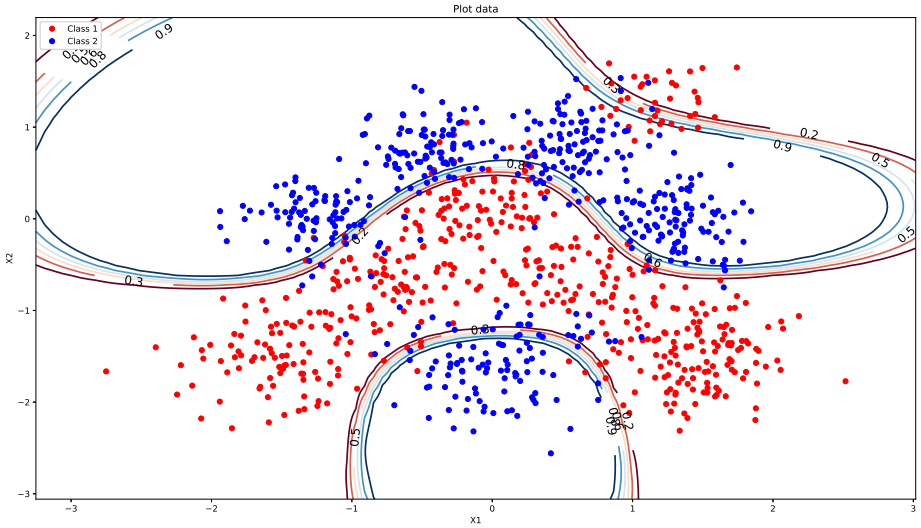


Figure 6: Prediction Visualisation for $l = 1$

l = 0.01

The final log-likelihood values for the training and test datasets were found to be $\mathbf{-0.10}$ and $\mathbf{-0.80}$ respectively. The confusion matrices for each of the datasets were also found.

$$\text{Training Data} = \begin{bmatrix} 1.00 & 0.00 \\ 0.00 & 1.00 \end{bmatrix} \qquad \text{Test Data} = \begin{bmatrix} 0.52 & 0.48 \\ 0.13 & 0.87 \end{bmatrix}$$

l = 0.1

The final log-likelihood values for the training and test datasets were found to be $\mathbf{-0.09}$ and $\mathbf{-0.39}$ respectively. The confusion matrices for each of the datasets were also found.

$$\text{Training Data} = \begin{bmatrix} 0.95 & 0.05 \\ 0.04 & 0.96 \end{bmatrix} \qquad \text{Test Data} = \begin{bmatrix} 0.83 & 0.17 \\ 0.14 & 0.86 \end{bmatrix}$$

l = 1

The final log-likelihood values for the training and test datasets were found to be $\mathbf{-0.43}$ and $\mathbf{-0.48}$ respectively. The confusion matrices for each of the datasets were also found.

$$\text{Training Data} = \begin{bmatrix} 0.88 & 0.12 \\ 0.08 & 0.92 \end{bmatrix} \qquad \text{Test Data} = \begin{bmatrix} 0.90 & 0.10 \\ 0.09 & 0.91 \end{bmatrix}$$

The hyper-parameter l has a large effect on the performance. When $l = 0.01$ significant overfitting is observed whereby every single training data point is classified correctly however the classifier performs poorly on the test data. When $l = 0.1$ the classifier performs well on the test data, classifying 85% of test data points correctly, and improvement upon the 67% observed with the linear classifier. Thus the RBF feature expansion has increased the performance of the logistic classifier. For $l = 1$ the average log-likelihood for the test data is worse than that for $l = 0.1$ however more test data points have been classified correctly. Based on the visualisations it appears the model becomes more general as l increases, which explains why the classifier performs better on test data for larger values of l. For the case where $l = 0.1$, small classification regions are observed around individual outlying data points which reduces the generality of the classifier.