Name: Chon Hei LAM CRSid: chl64 Experiment date: 22nd Feb 2019

3F8 Lab: Logistic classification on 2D data into two classes

1. Summary

A dataset of 1000 two-dimensional inputs and binary class outputs was given, and a logistic classifier was to be trained to model this dataset. First a linear class boundary classifier was experimented. Then the prediction performance was improved by adopting a different, non-linear feature expansion classifier. The reason behind the improvement was discussed, and the possibility of overfitting the nonlinear classifier was flagged.

2. Preparation exercises

2.1 Gradient of log-likelihood function

In a Logistic Classification model, the log-likelihood function $\mathcal{L}(\underline{w}) = \ln\left[p(\underline{y}|\boldsymbol{X}, \underline{w})\right]$ is to be maximised so as to obtain an optimal set of model parameters $\underline{w}$. Therefore, a general expression of its gradient with respect to $\underline{w}$ needs to be known. Starting from the given expression of $p\left(y^{(n)}\middle|\underline{\tilde{x}}^{(n)}\right)$, first obtain a simplified form of the log-likelihood function:

$$\mathcal{L}(\underline{w}) = \ln\left[\prod_{n=1}^{N} p\left(y^{(n)}\middle|\underline{\tilde{x}}^{(n)}\right)\right] = \sum_{n=1}^{N} \ln\left[p\left(y^{(n)}\middle|\underline{\tilde{x}}^{(n)}\right)\right]$$

$$= \sum_{n=1}^{N}\left[y^{(n)} \ln \sigma\left(\underline{w}^T\underline{\tilde{x}}^{(n)}\right) + \left(1-y^{(n)}\right)\ln\left(1-\sigma\left(\underline{w}^T\underline{\tilde{x}}^{(n)}\right)\right)\right]$$

Therefore, the gradient can be computed by making use of the differentiation chain rule, treating $y^{(n)}$ and $\underline{\tilde{x}}^{(n)}$ as constant, and applying the property of the logistic function $\frac{d\sigma(\cdot)}{dx} = \sigma(\cdot)[1-\sigma(\cdot)]$:

$$\frac{\partial}{\partial \underline{w}}\mathcal{L}(\underline{w}) = \sum_{n=1}^{N}\left\{y^{(n)} \times \frac{1}{\sigma(\cdot)} \times \sigma(\cdot)[1-\sigma(\cdot)] \times \underline{\tilde{x}}^{(n)} + \left(1-y^{(n)}\right) \times \frac{1}{1-\sigma(\cdot)} \times (-1)\sigma(\cdot)[1-\sigma(\cdot)] \times \underline{\tilde{x}}^{(n)}\right\}$$

$$= \sum_{n=1}^{N} \underline{\tilde{x}}^{(n)}\left\{y^{(n)} \times [1-\sigma(\cdot)] - \left(1-y^{(n)}\right)\sigma(\cdot)\right\} = \boxed{\sum_{n=1}^{N} \underline{\tilde{x}}^{(n)}\left\{y^{(n)} - \sigma\left(\underline{w}^T\underline{\tilde{x}}^{(n)}\right)\right\}}$$

2.2 Pseudo-code for gradient ascent method

Gradient ascent method for tuning $\underline{w}$ to maximise $\mathcal{L}(\underline{w})$ is an iteration of the following equation:

$$\underline{w}^{new} = \underline{w}^{old} + \alpha\frac{\partial\mathcal{L}(\underline{w})}{\partial \underline{w}}, \qquad where\ \alpha > 0\ is\ the\ "learning\ rate"$$

When computing the gradient from the summation $\sum_{n=1}^{N} \underline{\tilde{x}}^{(n)}\left\{y^{(n)} - \sigma\left(\underline{w}^T\underline{\tilde{x}}^{(n)}\right)\right\}$ through a for-loop, an equivalent matrix equation can be used:

$$\frac{\partial\mathcal{L}(\underline{w})}{\partial \underline{w}} = X^T\left[\underline{y} - \sigma(X\underline{w})\right]$$

where the n^{th} row in matrix X is $\underline{\tilde{x}}^{(n)}$ transposed, $\underline{y}$ is a column vector storing the class of each data point $y \in \{0,1\}$, and $\sigma(\cdot)$ operates elementwise to the column vector $X\underline{w}$.

Pseudo-code for tuning the parameters $\underline{w}$:

```
def gradient_ascent(X_train, y_train, w, α = 0.001, epochs = 100)
    for n in range(epochs):
        gradient = np.dot(X_train.T, (y_train − logistic(np.dot(X_train, w))))
        w = w + α * gradient
    return w
w_final = gradient_ascent(X_train, y_train, w_initial)
```

The learning rate α is chosen by trial and error, in attempt of obtain a high enough rate which increases the average log-likelihood of $\underline{w}$ smoothly over time. If α is too large, the log-likelihood will bound around the maximum and will not converge precisely.

3. Lab results and Discussions

3.1 Visualisation of the 2D dataset

Fig 1 shows the coordinate distribution of the 1000 data points and their corresponding classes. Since the points from one class are located in between the points from the other class, it can already be predicted that a classifier model with a linear class boundary, which was implemented in the lab, will not perform exceptionally well.

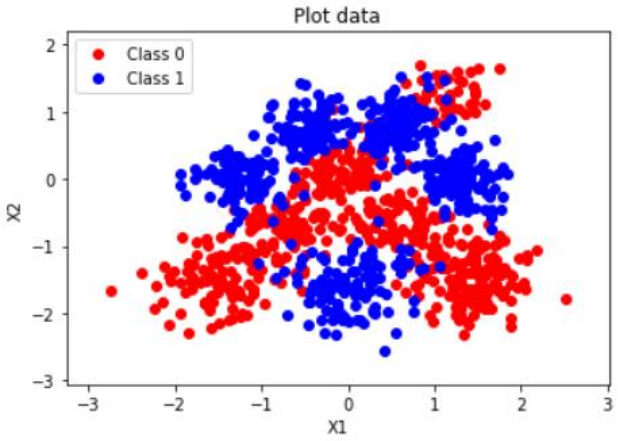

Fig 1. Graph of 2D coordinates for 1000 data-points classified into one of two classes.

3.2 Train classifier with linear class boundary

By unbiasing each datapoint from $\underline{x} = [x_1, x_2]$ to $\underline{\tilde{x}} = [1, x_1, x_2]$, the corresponding classifier parameters $\underline{w} = [w_0, w_1, w_2]$ will give a linear class boundary with the bias of $[x_1, x_2]$ considered. With learning rate $\alpha = 0.001$ and number of iterations (epochs) $= 100$, the gradient ascent method yields $\underline{w} = [0.3510, -0.0790, 0.8690]$.

Fig. 2 and 3 shows, for the training dataset and test dataset respectively, how average log-likelihood over all data points varies when the values of $\underline{w}$ are updated in each iteration. Note that the tuning of $\underline{w}$ was completely independent from the test set. The new $\underline{w}$ obtained from the training set in each step were simply used to compute the log-likelihood of the test set in Fig 3.

The final average log-likelihood for the train dataset was -0.614, when compared to -0.663 for the test set. Since the former is only slighter larger than the latter, the degree of overfitting is very small.

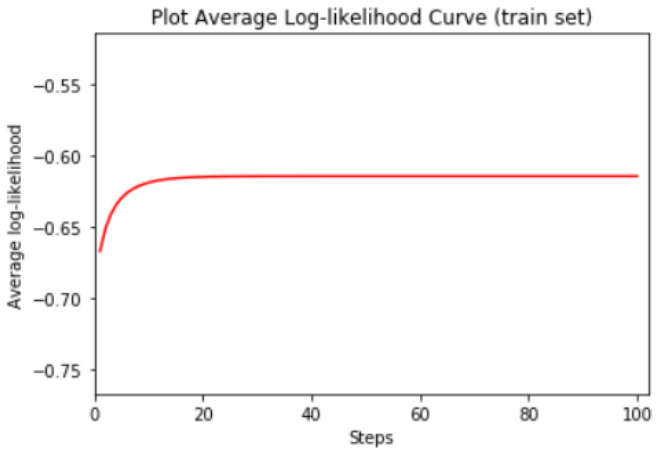

Fig 2 – Graph of average $\ln\left[p\left(y^{(n)}\middle|\underline{\tilde{x}}^{(n)}\right)\right]$ against the iteration step of gradient ascent for the training dataset. (Data points unbiased)

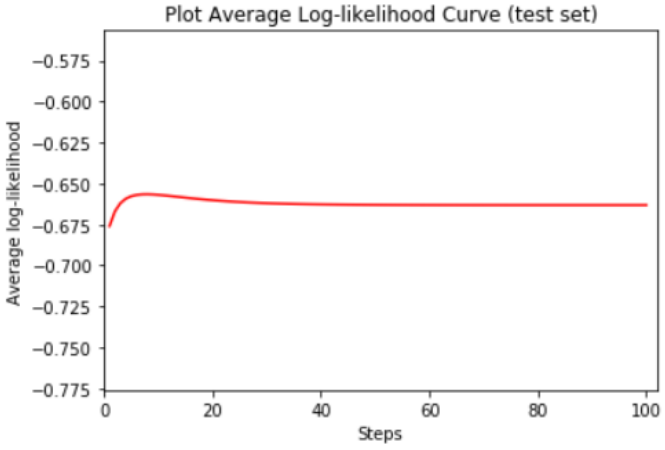

Fig 3 – Graph of average $\ln\left[p\left(y^{(n)}\middle|\underline{\tilde{x}}^{(n)}\right)\right]$ against the iteration step of gradient ascent for the test dataset. (Data points unbiased)

If probability contours for $p\left(y^{(n)}=1\middle|\underline{\tilde{x}}^{(n)},\underline{w}\right)$ are plotted upon the datapoint visualisation, as shown in Fig 4, it can be observed that this classifier with linear boundaries does not perform exceptional well (as predicted before). The blue class-1 points in the bottom region are assigned low probabilities of class-1, while the red class-0 points at the top are assigned high probabilities of class-1. A corresponding confusion matrix for this classifier was also created (Table 1), showing the moderate performance of the linear class boundary.

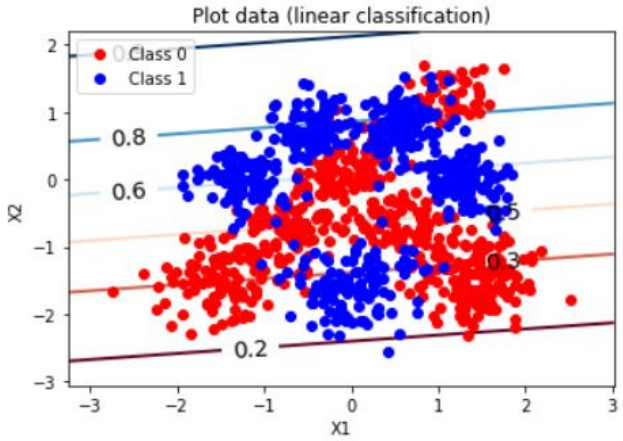

Fig 4 – linear probability contours overlapped on 2D data visualisation.

Central commands:

```
>> y_test_pred = np.dot(X_test, w)
>> thres = 0.5
>> y_test_pred = [i >= 0.5 for i in y_test_pred]
>> confusion_matrix(y_test, y_test_pred)
```

		Predicted class $\hat{y}$	
		0	1
True class y	0	79	22
	1	55	44

Table 1 – confusion matrix for logistic classification model with unbiased input data. (Central commands listed.)

3.3 Train classifier with radial basis functions (RBF)

To improve on the problem with linear class boundary, the features of each data point $\underline{x}^{(n)}$ (both training and test) can be expanded through the corresponding set of RBFs, centred on all training datapoints one by one. In our case where 800 out of 1000 2D-datapoints are for training, each $\underline{x}^{(n)}=[x_1,x_2]$ (both training and test) is expanded to possess $\underline{\tilde{x}}^{(n)}=[1,\tilde{x}_2,\ldots,\tilde{x}_{801}]$. Then similar to the linear class boundary case, the model parameters $\underline{w}$ are set to have the same dimension as each expanded datapoint. Finally the same gradient ascent algorithm with logistic function tunes $\underline{w}$.

```
>> X1_train = append_ones(expand_inputs(l, Xb_train, Xb_train))
>> X1_test = append_ones(expand_inputs(l, Xb_test, Xb_train))
```

Three different widths of RBF, $l = 1,\ 0.1,\ 0.01$, were experimented and their performance compared.

3.3.1 RBF width of $l = 1$ $(\alpha = 0.00005,\ no\ of\ epochs = 3000)$

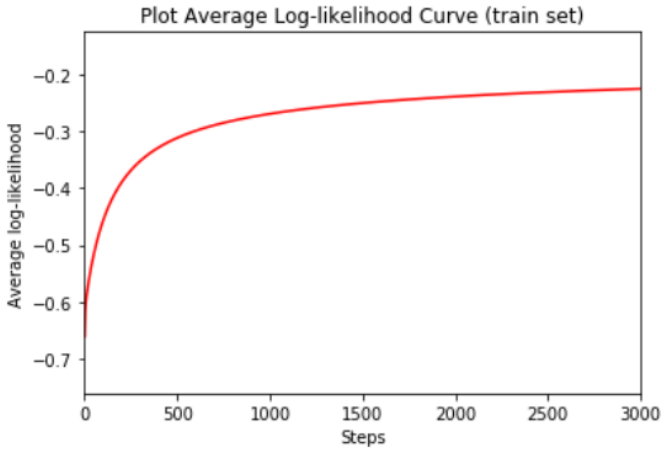

Fig 5 - Evolution of average $\ln\left[p\left(y^{(n)}\middle|\underline{\tilde{x}}^{(n)}\right)\right]$ for the training dataset (feature-expanded with $l = 1$).

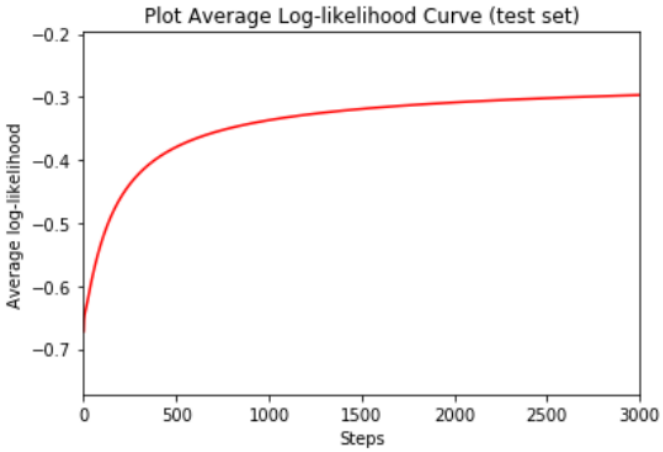

Fig 6 – Evolution of average $\ln\left[p\left(y^{(n)}\middle|\underline{\tilde{x}}^{(n)}\right)\right]$ for the test dataset (feature-expanded with $l = 1$).

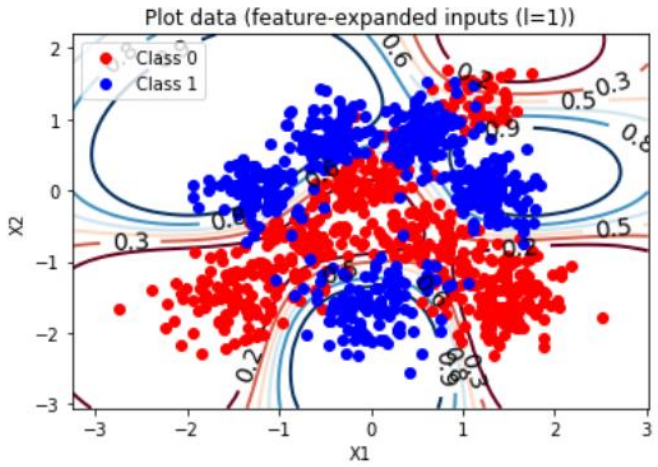

Fig 7 – Radial probability contours with width $l = 1$ overlapped on 2D data visualisation.

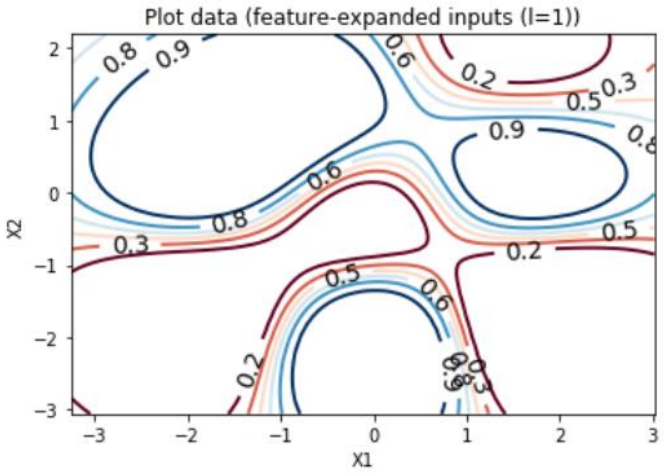

Fig 8 – Plot of radial probability contours with width $l = 1$ alone.

3.3.2 RBF width of $l = 0.1$ $(\alpha = 0.0007,\ no\ of\ epochs = 2000)$

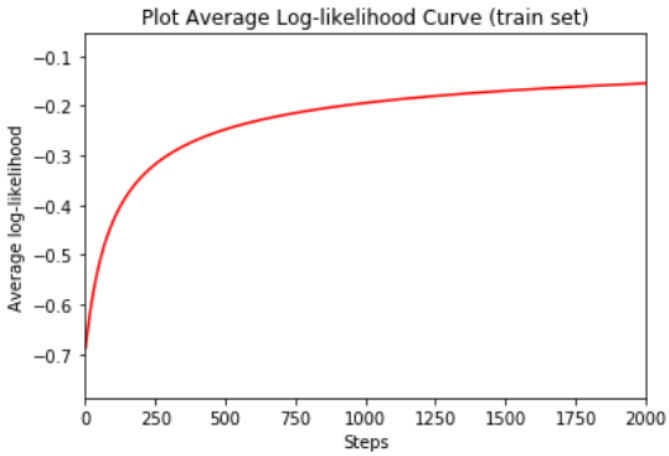

Fig 9 - Evolution of average $\ln\left[p\left(y^{(n)}\middle|\underline{\tilde{x}}^{(n)}\right)\right]$ for the training dataset (feature-expanded with $l = 0.1$).

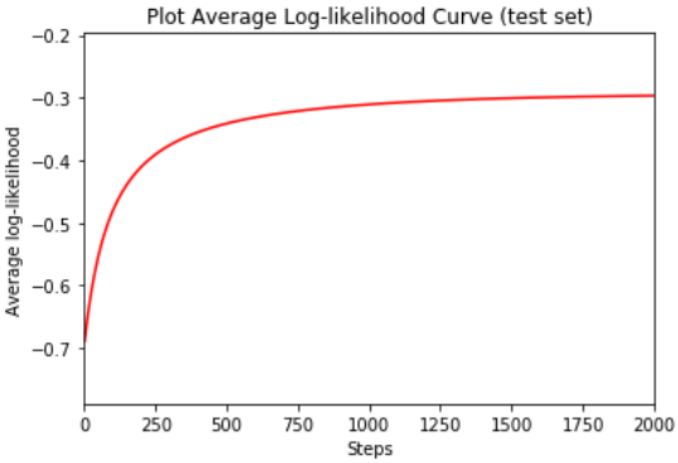

Fig 10 – Evolution of average $\ln\left[p\left(y^{(n)}\middle|\underline{\tilde{x}}^{(n)}\right)\right]$ for the test dataset (feature-expanded with $l = 0.1$).

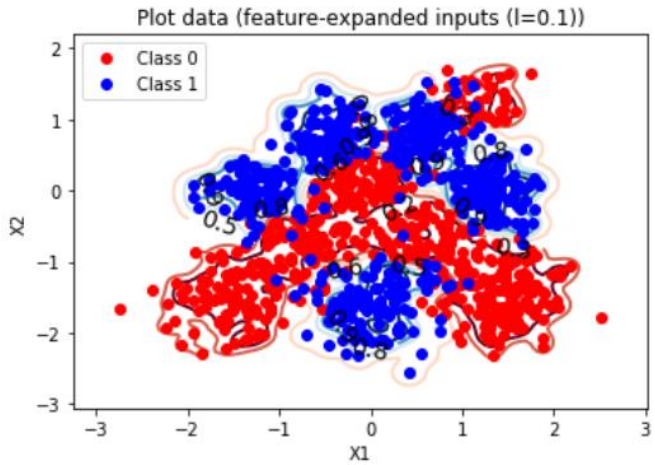

Fig 11 – Radial probability contours with width $l = 0.1$ overlapped on 2D data visualisation.

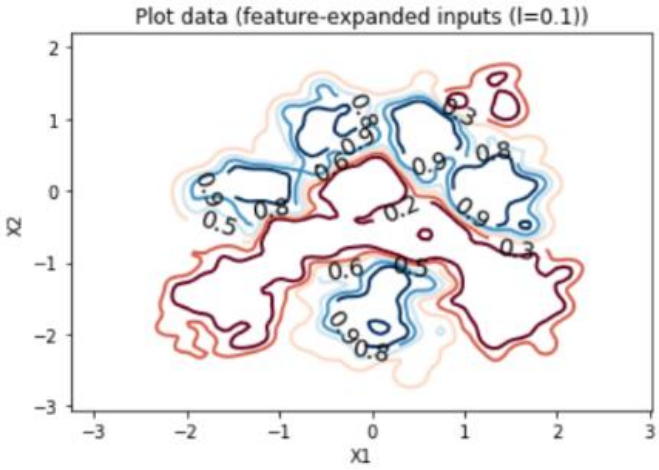

Fig 12 – Plot of radial probability contours with width $l = 0.1$ alone.

3.3.3 RBF width of $l = 0.01$ $(\alpha = 0.01,\ no\ of\ epochs = 2000)$

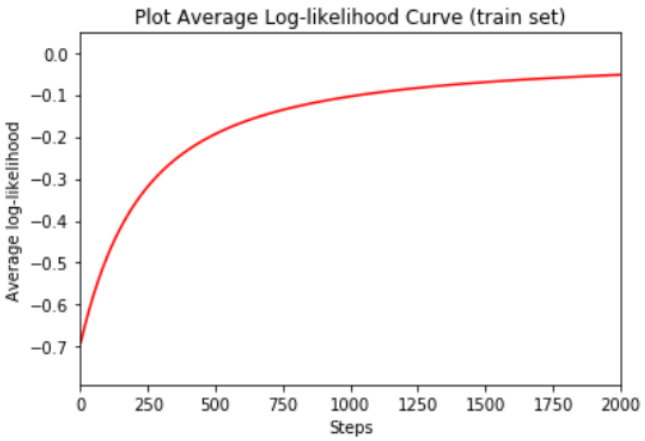

Fig 13 -Evolution of average $\ln\left[p\left(y^{(n)}\middle|\underline{\tilde{x}}^{(n)}\right)\right]$ for the training dataset (feature-expanded with $l = 0.01$).

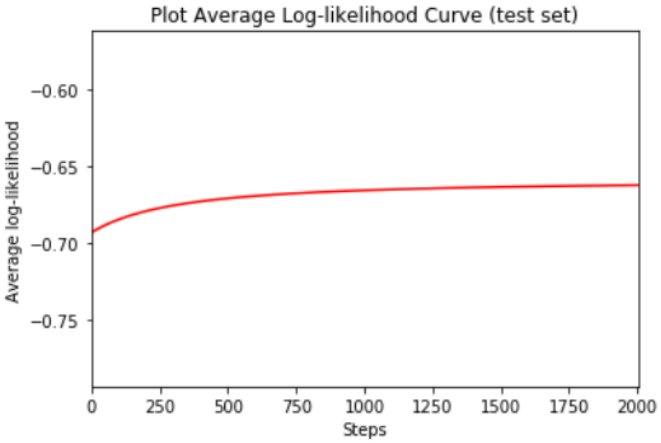

Fig 14 – Evolution of average $\ln\left[p\left(y^{(n)}\middle|\underline{\tilde{x}}^{(n)}\right)\right]$ for the test dataset (feature-expanded with $l = 0.01$).

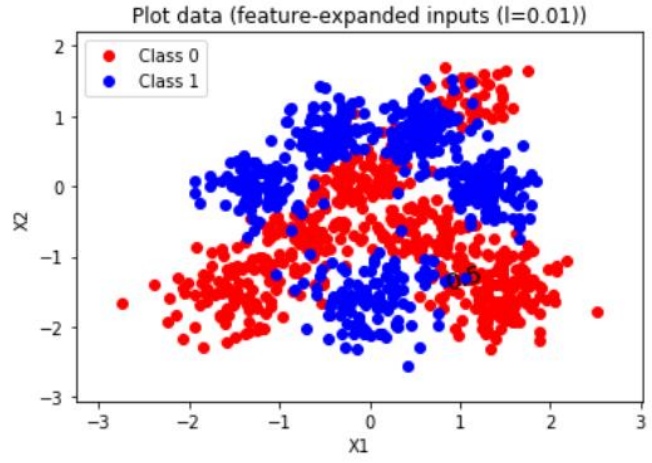

Fig 15 – Radial probability contours with width $l = 0.01$ overlapped on 2D data visualisation.

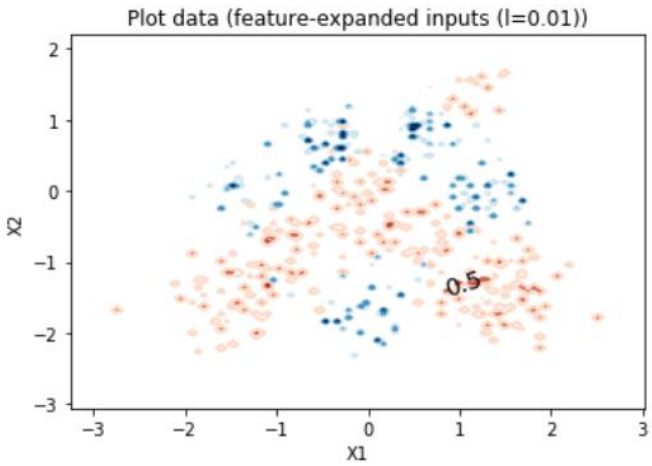

Fig 16 – Plot of radial probability contours with width $l = 0.01$ alone.

3.3.4 Comparison between the three RBF widths

Final average log-likelihood…	$l = 1$	$l = 0.1$	$l = 0.01$
… for train data point	-0.2251	-0.1548	-0.0524
… for test data point	-0.2968	-0.2967	-0.6624

Table 2 – Comparison between the final average log-likelihoods of $\underline{w}$ calculated upon train and test datasets, when $RBF\ width = 1, 0.1, 0.01$. (Values correspond to previous graphs on log-likelihood evolution, Fig 5&6, 9&10 and 13&14.)

From Table 2, it is clear that the final log-likelihood per training data point increases as the width of RBF is reduced. In contrast, that for the test data point remains very similar when l decreases from 1 to 0.1, but drops dramatically when l further decreases to 0.01. This phenomenon can be analysed together with the radial probability contour plots (Fig 8, 12 and 16). Reducing the RBF width causes the classifier model to fit the train data points better by making the probability contour spread less away from them. However this also means that the contour will cover less regions where the test data points lie, hence reducing the log-likelihood of the test points. This phenomenon is known as overfitting.

RBF width		$l = 1$		$l = 0.1$		$l = 0.01$	
Predicted class $\hat{y}$ →		0	1	0	1	0	1
True class y	0	89	12	93	8	101	0
	1	11	88	14	85	99	0

Table 3 – Three confusion matrices for the RBF classifiers with $l = 1, 0.1, 0.01$ respectively.

Table 3 combines the confusion matrices of all three cases together. Coherent to the final average log-likelihood values, the confusion matrices of $l = 1$ and 0.1 are very similar. When l decreases from 1 to 0.1, the probability contours match better the overall distribution shape of the datapoints, but those parts as the boundaries between the class clusters remain pretty much the same. This explains why $l = 0.1$ doesn't improve the confusion matrix or log-likelihood of the test data.

As mentioned, further reducing l to 0.01 causes overfitting. The confusion matrix for $l = 0.01$ shows that all test points are predicted as class 0, when nearly half of them actually belong to class 1. The reason for this is given by Fig 16. Since the high probability contours have shrunk so narrowly to just surround each of the train datapoints, the test points which obviously do not coincide with the train points will lie outside the high probability contours for class 1. Most if not all test points will be predicted as class 0 as a result.

4. Conclusion

There are at least two ways to train a logistic classifier, namely by employing a linear class boundary or multiple radial boundaries. The former only works well when the datapoints from one class all cluster away from those of the other class. In contrast the latter approach can adapt more unstructured cluster distribution and hence generally results in a better model. One caveat for using the radial boundary classifier is to avoid setting the radius l to be too small, which will lead to overfitting and damage the classifier performance.