3F8: Inference
Short Lab Report

Minghong Zhao, Peterhouse, mz492

February 8, 2026

Abstract

We implement logistic regression for binary classification on a 2D dataset and evaluate it using average train/test log-likelihood and a test confusion matrix. We then apply a Gaussian RBF feature expansion with widths $l \in \{0.01, 0.1, 1\}$ and show that it improves performance by enabling non-linear decision boundaries, while l and the learning rate control generalisation and numerical stability.

1 Introduction

Logistic regression is a probabilistic classifier whose decision boundary is linear in the input space, so it is expected to struggle on non-linearly separable data. In this report we derive and implement gradient ascent for maximum-likelihood training, evaluate performance using learning curves, average train/test log-likelihood and confusion matrices, and then improve the model using a Gaussian RBF feature expansion centred on training points. By varying the RBF width $l \in \{0.01, 0.1, 1\}$ (with tuned learning rates), we illustrate the trade-off between overly local features, generalisation, and optimisation stability.

2 Exercise a)

For the logistic classification, the observed values, outcome, n-th variable and parameters are given as

$$\mathbf{X} \in \mathbb{R}^{N\times D}, \qquad \mathbf{y} \in \{0,1\}^N, \qquad \tilde{\mathbf{x}}^{(n)} = (1, \mathbf{x}^{(n)}), \qquad \boldsymbol{\beta} \in \mathbb{R}^{D+1}.$$

By definition of logistic model and the binary outcomes of $y^{(n)}$,

$$P\big(y^{(n)} = 1 \mid \tilde{\mathbf{x}}^{(n)}, \boldsymbol{\beta}\big) = \frac{1}{1+\exp\Big(-\boldsymbol{\beta}^\top \tilde{\mathbf{x}}^{(n)}\Big)} = \sigma\Big(\boldsymbol{\beta}^\top \tilde{\mathbf{x}}^{(n)}\Big).$$

$$P\big(y^{(n)} = 0 \mid \tilde{\mathbf{x}}^{(n)}, \boldsymbol{\beta}\big) = 1 - \sigma\Big(\boldsymbol{\beta}^\top \tilde{\mathbf{x}}^{(n)}\Big) = \sigma\Big(-\boldsymbol{\beta}^\top \tilde{\mathbf{x}}^{(n)}\Big).$$

These two cases can be written compactly using the Bernoulli probability mass function as

$$P\big(y^{(n)} \mid \tilde{\mathbf{x}}^{(n)}, \boldsymbol{\beta}\big) = \sigma\Big(\boldsymbol{\beta}^\top \tilde{\mathbf{x}}^{(n)}\Big)^{y^{(n)}} \Big(1 - \sigma\Big(\boldsymbol{\beta}^\top \tilde{\mathbf{x}}^{(n)}\Big)\Big)^{1-y^{(n)}}.$$

Since the observations are drawn i.i.d. independently, the likelihood of the full dataset is the product of the individual Bernoulli likelihoods,

$$P(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\beta}) = \prod_{n=1}^{N} P\big(y^{(n)} \mid \tilde{\mathbf{x}}^{(n)}, \boldsymbol{\beta}\big) = \prod_{n=1}^{N} \sigma\Big(\boldsymbol{\beta}^\top \tilde{\mathbf{x}}^{(n)}\Big)^{y^{(n)}} \Big(1 - \sigma\Big(\boldsymbol{\beta}^\top \tilde{\mathbf{x}}^{(n)}\Big)\Big)^{1-y^{(n)}}.$$

Taking the logarithm yields the log-likelihood function,

$$\mathcal{L}(\boldsymbol{\beta}) = \log P(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\beta}) = \sum_{n=1}^{N} \left[y^{(n)} \log \sigma\left(\boldsymbol{\beta}^\top \tilde{\mathbf{x}}^{(n)}\right) + (1 - y^{(n)}) \log\left(1 - \sigma\left(\boldsymbol{\beta}^\top \tilde{\mathbf{x}}^{(n)}\right)\right)\right].$$

Let

$$z = \boldsymbol{\beta}^\top \tilde{\mathbf{x}},$$

then let

$$\ell_n(z(\boldsymbol{\beta})) = y \log \sigma(z) + (1 - y) \log(1 - \sigma(z))$$

where $\sigma(z) = (1 + e^{-z})^{-1}$, then

$$\frac{d\sigma}{dz} = -1 \cdot (1 + e^{-z})^{-2} \cdot (-e^{-z}) = \frac{e^{-z}}{(1 + e^{-z})^2} = \left(\frac{1}{1 + e^{-z}}\right)\left(\frac{e^{-z}}{1 + e^{-z}}\right) = \sigma(z)\,[1 - \sigma(z)],$$

$$\frac{d}{dz} \log \sigma(z) = \frac{1}{\sigma(z)} \cdot \sigma(z)(1 - \sigma(z)) = 1 - \sigma(z)$$

$$\frac{d}{dz} \log(1 - \sigma(z)) = \frac{-1}{1 - \sigma(z)} \cdot \sigma(z)(1 - \sigma(z)) = -\sigma(z)$$

$$\frac{d\ell_n}{dz} = y(1 - \sigma(z)) + (1 - y)(-\sigma(z)) = y - \sigma(z)$$

Therefore the gradient of the log-likelihood function is

$$\frac{\partial \mathcal{L}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \sum_{n=1}^{N} \frac{d\ell_n}{dz} \cdot \frac{dz}{d\boldsymbol{\beta}} = \sum_{n=1}^{N} (y - \sigma(z))\, \tilde{\mathbf{x}} = \sum_{n=1}^{N} \left(y^{(n)} - \sigma\left(\boldsymbol{\beta}^\top \tilde{\mathbf{x}}^{(n)}\right)\right) \tilde{\mathbf{x}}^{(n)}.$$

3 Exercise b)

Different from *linear regression*, we can use maximum likelihood estimation to estimate the parameters $\boldsymbol{\beta}$, however, in *logistic regression*, we cannot as it is a nonlinear log-likelihood gradient, hence the gradient ascent method is chosen.

Before starting the gradient ascent algorithm to maximise the log-likelihood, we need to confirm that the objective function is *concave*, in logistic regression, this is guaranteed by the linear combination of both concave functions $\log \sigma(z)$ and $\log(1 - \sigma(z))$.

The following is a valid gradient ascent algorithm pseudocode for the optimisation problem

$$\begin{aligned} \text{maximise}_{\boldsymbol{\beta} \in \mathbb{R}^{D+1}} \quad & \sum_{n=1}^{N} \left[y^{(n)} \log \sigma\left(\boldsymbol{\beta}^\top \tilde{\mathbf{x}}^{(n)}\right) + \left(1 - y^{(n)}\right) \log\left(1 - \sigma\left(\boldsymbol{\beta}^\top \tilde{\mathbf{x}}^{(n)}\right)\right)\right] \\ \text{subject to} \quad & \boldsymbol{\beta} \in \mathbb{R}^{D+1}. \end{aligned}$$

Function estimate_parameters

Input:

Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$
Label vector $\mathbf{y} \in \mathbb{R}^{N \times 1}$

Output:

Parameter vector $\boldsymbol{\beta} \in \mathbb{R}^{D \times 1}$

Initialise $\boldsymbol{\beta}^{(\text{old})} \leftarrow \mathbf{0}$
Choose learning rate η

Set maximum number of iterations T

For $t = 1$ **to** T **do**

Compute gradient $\nabla\mathcal{L}\left(\boldsymbol{\beta}^{(\text{old})}\right)$ using $\mathbf{X}$ and $\mathbf{y}$

$\boldsymbol{\beta}^{(\text{new})} \leftarrow \boldsymbol{\beta}^{(\text{old})} + \eta\nabla\mathcal{L}\left(\boldsymbol{\beta}^{(\text{old})}\right)$

$\boldsymbol{\beta}^{(\text{old})} \leftarrow \boldsymbol{\beta}^{(\text{new})}$

End for

Return $\boldsymbol{\beta}^{(\text{new})}$

The learning rate parameter η is chosen to ensure stable convergence of the gradient ascent algorithm. In practice, several values of η can be tested by monitoring the evolution of the log-likelihood during training. If the log-likelihood decreases or oscillates, the learning rate is reduced; if convergence is excessively slow, the learning rate is increased.

4 Exercise c)

In this exercise we visualise the dataset in the two-dimensional input space displaying each datapoint's class label. The dataset is visualised in Figure 1. From the figure, the two classes are highly interleaved and not linearly separable in the input space. Consequently, a classifier with a single linear decision boundary is expected to achieve limited performance, as it cannot separate the classes without incurring a substantial number of misclassifications.

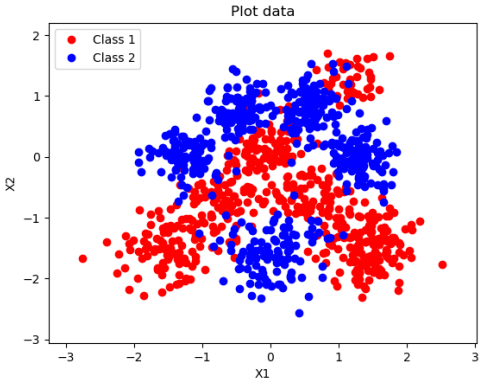

Figure 1: Visualisation of the data.

5 Exercise d)

The data are randomly split into a training set of 800 data points and a test set of 200 data points. The pseudocode derived in Exercise a) is implemented in Python as shown below.

```
def fit_w(X_tilde_train, y_train, X_tilde_test, y_test, n_steps, alpha):
    w = np.random.randn(X_tilde_train.shape[1])
    ll_train = np.zeros(n_steps)
    ll_test = np.zeros(n_steps)

    for i in range(n_steps):
        sigmoid_value = predict(X_tilde_train, w)
        gradient = np.dot(X_tilde_train.T, y_train - sigmoid_value)
```

```
        w = w + alpha * gradient

        ll_train[i] = compute_average_ll(X_tilde_train, y_train, w)
        ll_test[i] = compute_average_ll(X_tilde_test, y_test, w)

    return w, ll_train, ll_test
```

We then train the logistic regression classifier using this implementation. Several values of the learning rate η were tested empirically. Larger values of η were observed to lead to unstable optimisation behaviour, while smaller values resulted in smoother convergence. A learning rate of $\eta = 0.001$ was therefore selected for the experiments reported here.

The average log-likelihood values on the training and test sets as the optimisation proceeds are shown in Figure 3. The two curves converge rapidly and remain close to each other, indicating that the model reaches a stable solution without severe overfitting.

Figure 4 shows the contours of the class predictive probabilities overlaid on the data. The contours correspond to level sets of the predicted probability $p(y = 1 \mid \mathbf{x})$. As expected for a linear logistic regression model, the contours form approximately parallel lines, illustrating that the resulting decision boundary is linear and therefore limited in its ability to capture the non-linear structure of the data.

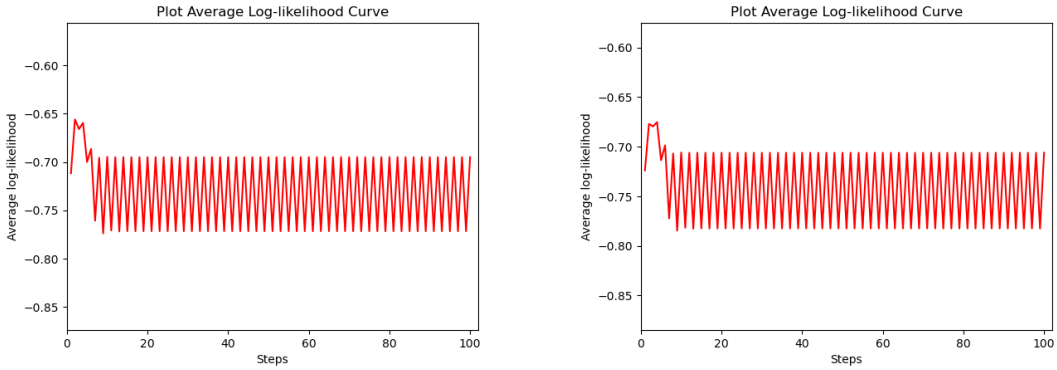

Figure 2: Learning curves showing the average log-likelihood on the training (left) and test (right) datasets, obtained with learning rate $\eta = 0.01$.

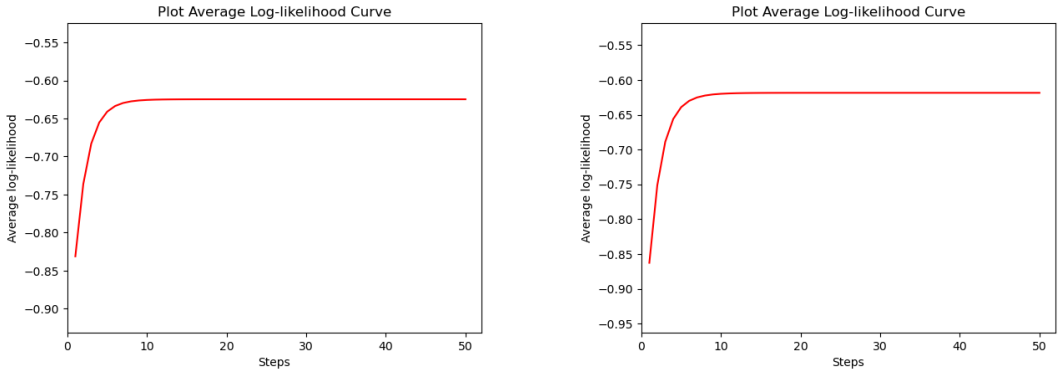

Figure 3: Learning curves showing the average log-likelihood on the training (left) and test (right) datasets, obtained with learning rate $\eta = 0.001$.

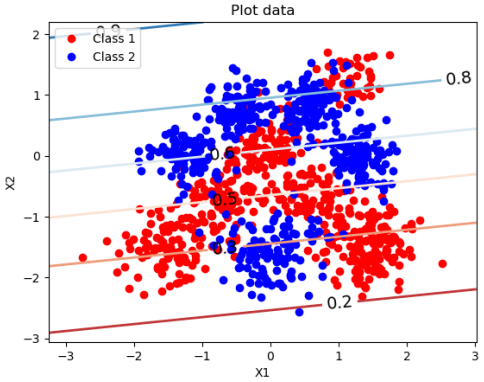

Figure 4: Visualisation of the contours of the class predictive probabilities.

6 Exercise e)

The final average training and test log-likelihoods per datapoint are shown in Table 1. The 2×2 confusion matrix on the test set is shown in Table 2. By analysing this table, we conclude that the linear logistic regression model is able to perform meaningful classification, but still makes a substantial number of misclassifications, reflecting the limitations of a linear decision boundary.

Avg. Train ll	**Avg. Test ll**
-0.6247	-0.6185

Table 1: Average training and test log-likelihoods.

		$\hat{y}$	
		0	1
y	0	0.380	0.150
	1	0.145	0.325

Table 2: Confusion matrix on the test set.

7 Exercise f)

We now expand the inputs through a set of Gaussian radial basis functions centred on the training datapoints. We consider widths $l = \{0.01, 0.1, 1\}$ for the basis functions. We fix the learning rate parameter to be $\eta = \{0.01, 0.01, 0.001\}$ for each $l = \{0.01, 0.1, 1\}$, respectively. Figure 5 displays the visualisation of the contours of the resulting class predictive probabilities on top of the data for each choice of $l = \{0.01, 0.1, 1\}$.

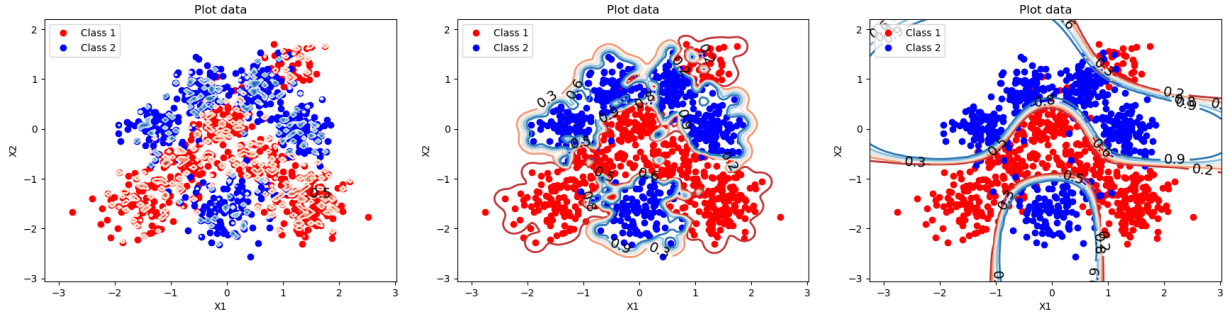

Figure 5: Visualisation of the contours of the class predictive probabilities for $l = 0.01$ (left), $l = 0.1$ (middle), $l = 1$ (right).

8 Exercise g)

The final final training and test log-likelihoods per datapoint obtained for each setting of $l = \{0.01, 0.1, 1\}$ are shown in tables 3, 4 and 5. These results indicate that the choice of l has a strong effect on generalisation: $l = 0.01$ achieves the best training log-likelihood but the worst test log-likelihood, suggesting that overly narrow basis functions do not generalise well. For $l = 1$, NaN values were observed during optimisation for larger learning rates due to numerical saturation of predicted probabilities; this was fixed by reducing the learning rate, and the final stable results are reported in Table 5.

The 2×2 confusion matrices for the three models trained with $l = \{0.01, 0.1, 1\}$ are show in tables 6, 7 and 8. After analysing these matrices, we can say that $l = 0.01$ strongly biases the classifier towards predicting $\hat{y} = 0$ (very low TP and many false negatives), while $l = 0.1$ and $l = 1$ yield a much more balanced classifier with substantially higher TP.

When we compare these results to those obtained using the original inputs we conclude that the RBF feature expansion significantly improves performance on this non-linear dataset by enabling non-linear decision boundaries that a linear classifier in the original input space cannot represent.

Avg. Train ll	**Avg. Test ll**
-0.00991491	-0.68622012

Table 3: Results for $l = 0.01$

Avg. Train ll	**Avg. Test ll**
-0.04202689	-0.60242385

Table 4: Results for $l = 0.1$

Avg. Train ll	**Avg. Test ll**
-0.30586089	-0.2511782

Table 5: Results for $l = 1$

		$\hat{y}$	
		0	1
y	0	0.465	0.005
	1	0.445	0.085

Table 6: Conf. matrix $l = 0.01$.

		$\hat{y}$	
		0	1
y	0	0.425	0.045
	1	0.085	0.445

Table 7: Conf. matrix $l = 0.1$.

		$\hat{y}$	
		0	1
y	0	0.440	0.030
	1	0.045	0.485

Table 8: Conf. matrix $l = 1$.

9 Conclusions

Logistic regression on the original inputs performs meaningful classification but is limited by its linear decision boundary, consistent with the strongly interleaved structure of the data and the baseline confusion matrix / test log-likelihood.

Applying a Gaussian RBF feature expansion enables non-linear decision boundaries and substantially improves predictive performance. The RBF width $l \in \{0.01, 0.1, 1\}$ strongly affects generalisation: $l = 0.01$ is overly local and generalises poorly (high training log-likelihood but worse test performance with many false negatives), whereas $l = 0.1$ and $l = 1$ produce more balanced confusion matrices and higher test performance; among these, $l = 1$ performed best overall in our experiments.

A practical limitation is numerical and optimisation stability: with larger l and aggressive learning rates, probability saturation can lead to $\log(0)$ and NaNs in the log-likelihood, which we avoided by reducing the learning rate (we used $\eta = \{0.01, 0.01, 0.001\}$ for $l = \{0.01, 0.1, 1\}$). Finally, training log-likelihood alone is not a reliable indicator of generalisation, so hyperparameters should be chosen based on test/validation performance; regularisation and more numerically stable log-likelihood implementations (e.g. clipping or stable log-sigmoid) could further improve robustness.