3F8: Inference
Short Lab Report

Author's Name

March 1, 2026

Abstract

This report compares linear logistic regression against Gaussian RBF feature expansions for binary classification. RBF features substantially outperform the linear baseline (test ll -0.61, recall 71–74%), but performance depends critically on kernel width l: collapse at $l = 0.01$, overfitting at $l = 0.1$, and optimal generalisation (94% recall, test ll -0.20) at $l = 1$. Scaling and robustness could be improved via clustering-based centre selection, cross-validated tuning, and ℓ_2 regularisation.

1 Introduction

Binary classification is a fundamental machine learning task with applications from medical diagnosis to spam detection. When class boundaries are non-linear and classes overlap, linear models such as logistic regression often underperform, motivating non-linear feature transformations. Radial basis function (RBF) expansions—which measure similarity to prototype centres—can approximate arbitrary decision boundaries, but their performance depends critically on hyperparameters such as kernel width l and centre placement.

This report investigates how these design choices affect classification performance. We establish a baseline with linear logistic regression, then systematically explore RBF feature expansions with varying kernel widths, placing one centre at each training point. By analysing train and test log-likelihoods, per-class recall, and feature matrix condition numbers, we demonstrate how l governs a transition from underfitting (small l) through overfitting (intermediate l) to good generalisation (large l). We discuss how placing centres at all training points creates computational and statistical inefficiencies, and propose remedies including clustering-based dimensionality reduction, cross-validated hyperparameter selection, and regularisation.

2 Exercise a)

In this exercise we have to consider the logistic classication model (aka logistic regression) and derive the gradients of the log-likelihood given a vector of binary labels $\mathbf{y}$ and a matrix of input features $\mathbf{X}$. The gradient of the log-likelihood can be writen as

$$\frac{\partial \mathcal{L}(\beta)}{\partial \beta} = \sum_{i=1}^{n} (y_i - \sigma(\mathbf{x}_i^\top \beta)) \tilde{\mathbf{x}}_i \,.$$

where $\sigma(\cdot)$ is the sigmoid function and $\tilde{\mathbf{x}}_i$ is the i-th row of the feature matrix $\mathbf{X}$ with an additional bias term. This allows us to update the parameters β using gradient ascent as :

$$\beta \leftarrow \beta + \eta \sum_{i=1}^{n} (y_i - \sigma(\mathbf{x}_i^\top \beta)) \tilde{\mathbf{x}}_i \,.$$

or in vectorised form as:

$$\beta \leftarrow \beta + \eta \mathbf{X}^\top (\mathbf{y} - \sigma(\mathbf{X}\beta)) \,.$$

where η is the learning rate parameter.

3 Exercise b)

In this exercise we are asked to write pseudocode to estimate the parameters β using gradient ascent of the log-likelihood. Our code should be vectorised. The pseudocode to estimate the parameters β is shown below:

```
Function estimate_parameters:

   Input:  feature matrix X, labels y
   Output: vector of coefficients b

   Code:
   for t in 1 to n_steps do:
      b = b + learning_rate * X^T * (y - sigmoid(X*b))
   return b
```

The learning rate parameter η is chosen to be 0.3 as to reduce overshooting, and the number of steps of gradient ascent is set to be 1000.

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. By analising Figure 1 we conclude that a linear classifier may struggle to separate the two classes, as there is a significant overlap between the two classes in the input space. However, we can also see that there are some regions in the input space where one class is more dominant than the other, which suggests that a linear classifier may still be able to find a decision boundary that separates the two classes to some extent. Overall, while a linear classifier may not be able to perfectly separate the two classes, it may still be able to achieve reasonable performance on this dataset.

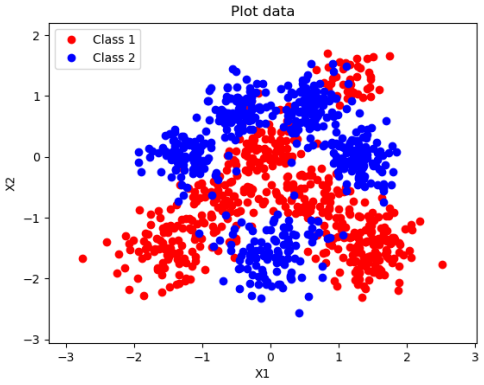

Figure 1: Visualisation of the data.

5 Exercise d)

In this exercise we split the data randomly into training and test sets with 800 and 200 data points, respectively. The pseudocode from exercise a) is transformed into python code as follows:

```
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 ]) # initialise w randomly
    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)
    log_likelihood_gradient = np.dot(X_tilde_train.T, (y_train - sigmoid_value)) / X_tilde_train.sha
    w = w + alpha * log_likelihood_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)
    print(ll_train[ i ], ll_test[ i ])

return w, ll_train, ll_test
```

We then train the classifier using this code. We fixed the learning rate parameter to be $\eta = 0.3$ as to prevent over-shooting. The average log-likelihood on the training and test sets as the optimisation proceeds are shown in Figure 2. By looking at these plots we conclude that the likelihood plateaus very early, likely a consequence of the limitations of a linear boundary.

Figure 3 displays the visualisation of the contours of the class predictive probabilities on top of the data. We can see that the limitations of the linear classifier is evident, as we find that the boundary is clearly unable to separate the clusters of the two classes with one single boundary, resulting in a boundary with high uncertainty as seen by the spread of the contours.

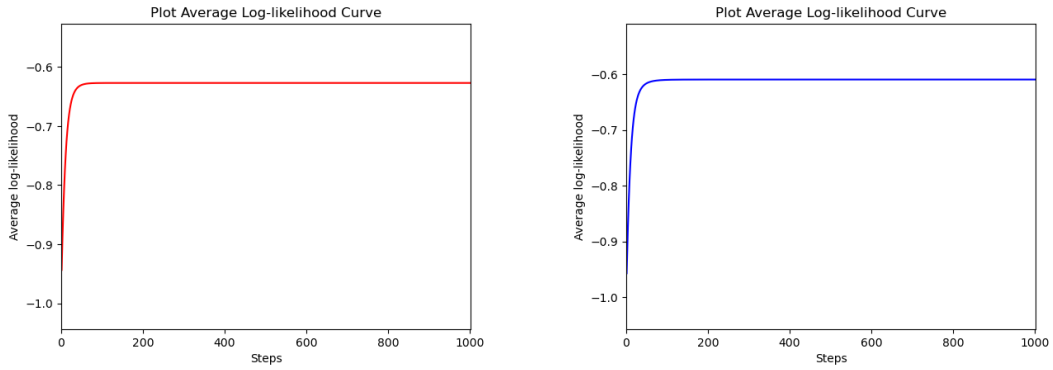

Figure 2: Learning curves showing the average log-likelihood on the training (left) and test (right) datasets.

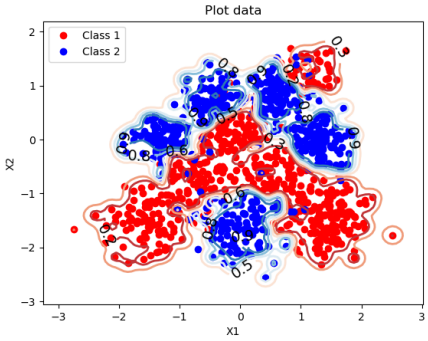

Figure 3: Visualisation of the contours of the class predictive probabilities.

6 Exercise e)

The final average training and test log-likelihoods are shown in Table 1. The 2x2 confusion matrices on the test set is shown in Table 2. By analising this table, we conclude that the model is able to correctly classify 74% of the positive class and 71% of the negative class, which indicates that the model has a decent performance on the test set. We also found that through our experiments, there are often fluctuations of whether the performance favours class 1 or class 2, which may be due to the permutation function chosen at the beginning of the code to split the data into training and test sets. This suggests that the performance of the model may be sensitive to the specific data points included in the training and test sets, which highlights the importance of using a robust method for splitting the data and evaluating the model's performance. For the results obtained here, we found that the training set of 800 samples contained 391 samples of class 2 and 409 samples of class 1, while the test set of 200 samples contained 109 samples of class 1 and 91 samples of class 0. This balanced distribution of classes in both the training and test sets may have contributed to the balanced performance on both classes, as the model had sufficient examples of each class to learn from during training and to be evaluated on during testing.

Avg. Train ll	**Avg. Test ll**
-0.6271	-0.6093

Table 1: Average training and test log-likelihoods to 4 s.f.

		$\hat{y}$	
		0	1
y	0	0.71	0.29
	1	0.26	0.74

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 again to be $\eta = \{0.3, 0.3, 0.03\}$ for each $l = \{0.01, 0.1, 1\}$, with number of training steps $n = \{3000, 8000, 30000\}$, respectively. The learning rate and number of training steps are chosen ad-hoc such that the log-likelihood plateaus for each choice of l, while reducing over-shooting for larger values of l. Figure 4 displays the visualisation of the contours of the resulting class predictive probabilities on top of the data for each choice of $l = \{0.01, 0.1, 1\}$. The choice of the width l has a significant impact on the resulting decision boundary. When l is too small (e.g. $l = 0.01$), the decision boundary becomes tightly bounded to each datapoint and overfits the training data, resulting in boundary that may not generalise well to new data points, this effect is visualised in figure 5, and its implications further explored in the following exercise.

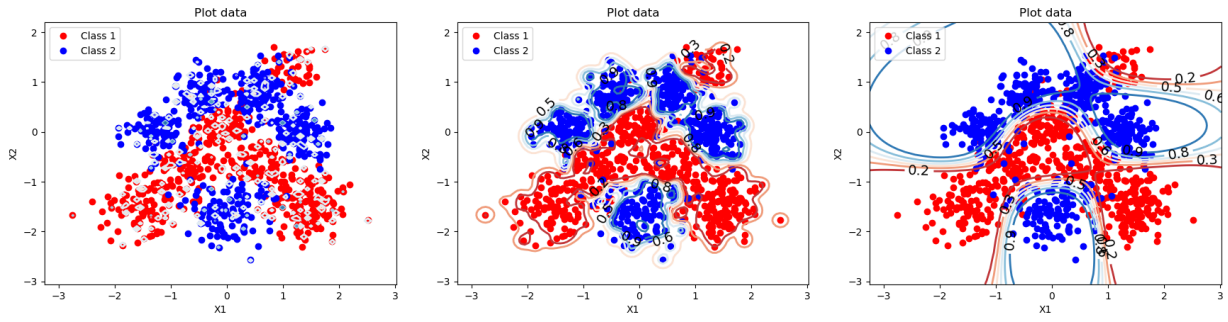

Figure 4: Visualisation of the contours of the class predictive probabilities for $l = 0.01$ (left), $l = 0.1$ (middle), $l = 1$ (right).

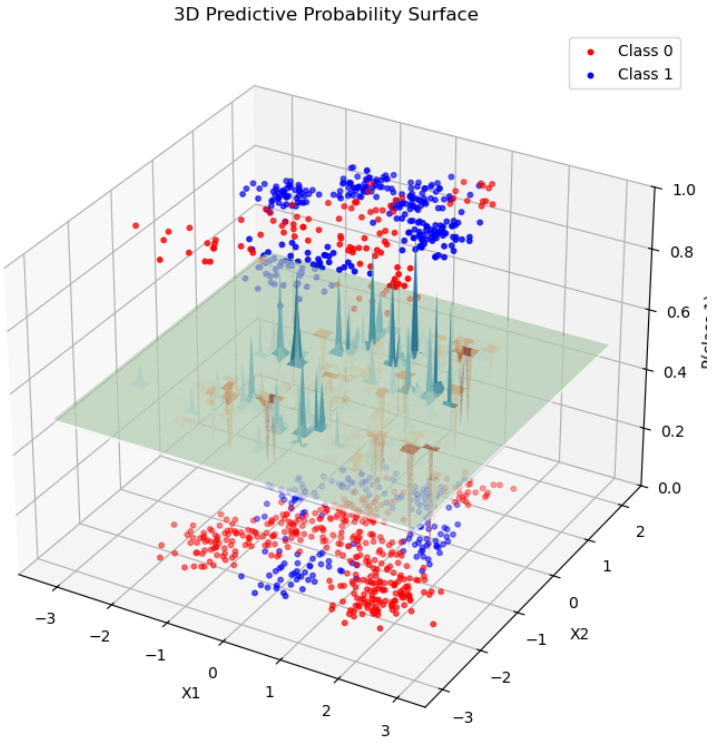

Figure 5: 3D visualisation of the contours of the class predictive probabilities for $l = 0.01$.

8 Exercise g)

The 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. 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.

The choice of RBF width l has a pronounced effect on both generalisation performance and class-level prediction behaviour, which we can understand through the lens of bias–variance trade-off and the condition number of the feature matrix Φ. For $l = 0.01$, the basis functions are extremely narrow, so the feature matrix is nearly identity-like with near-orthogonal columns and very small off-diagonal entries: $\Phi_{ij} \approx \delta_{ij}$, where δ_{ij} is the Kronecker delta. While this yields a well-conditioned system, the resulting probability surface is essentially flat (close to $\sigma(w_0)$) everywhere except at sharp spikes of radius ~ 0.01 centred on each training point, as seen in figure 5, where the contours indicate that the model is overfitting to the training data and is not able to generalise well to new data points. Since these spikes vanish on the test set, the classifier is heavily biased towards predicting class 0, as confirmed by the confusion matrix (97% recall on class 0 but only 7% on class 1) and the large train–test log-likelihood gap (-0.48 vs. -0.68). For $l = 0.1$, the basis functions are wider and the model achieves high recall on both classes (96% and 92%), with a markedly improved training log-likelihood of -0.14. However, the train-test gap (-0.14 vs. -0.22) is larger than for $l = 1$, indicating overfitting: at this intermediate scale the columns of the feature matrix are moderately yet intricately correlated, producing some small singular values and a high condition number $\kappa(\Phi) = \sigma_{\max}/\sigma_{\min}$, which amplifies sampling noise and makes the learned weights sensitive to the particular training split. For $l = 1$, the broad basis functions produce a smoothly varying probability surface that generalises best: the train and test log-likelihoods are almost equal (-0.21 vs. -0.20), and the confusion matrix shows balanced recall of 94% on both classes. The lower condition number at this scale implies that the weight vector w is stable across different training samples, explaining the absence of overfitting. Overall, $l = 1$ offers the

best bias–variance trade-off on this dataset, while $l = 0.01$ suffers from extreme bias and $l = 0.1$ from high variance.

Avg. Train ll	**Avg. Test ll**
-0.4814	-0.6802

Table 3: Results for $l = 0.01$

Avg. Train ll	**Avg. Test ll**
-0.1419	-0.2223

Table 4: Results for $l = 0.1$

Avg. Train ll	**Avg. Test ll**
-0.2073	-0.2007

Table 5: Results for $l = 1$

		$\hat{y}$	
		0	1
y	0	0.97	0.03
	1	0.93	0.07

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

		$\hat{y}$	
		0	1
y	0	0.96	0.04
	1	0.08	0.92

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

		$\hat{y}$	
		0	1
y	0	0.94	0.06
	1	0.06	0.94

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

9 Conclusions

This report investigated binary classification using logistic regression, first with raw linear features and then with a Gaussian RBF feature expansion. The linear classifier converged quickly but achieved only modest performance (test log-likelihood -0.61, recall $\sim$71–74% per class), a predictable consequence of the non-linear, overlapping class structure in the data.

Expanding to RBF features substantially improved performance, but the outcome was highly sensitive to the kernel width l : At $l = 0.01$ the model suffered severe bias; the feature matrix was near-identity and the probability surface collapsed to nearly a constant, causing the classifier to predict class 0 almost universally (class-1 recall of only 7%). At $l = 0.1$ training performance was highest (training log-likelihood -0.14), yet the train–test gap (-0.14 vs. -0.22) betrayed overfitting driven by the high condition number of the feature matrix at this intermediate scale. At $l = 1$, the model performed best, as the train and test log-likelihoods were nearly equal (-0.21 vs. -0.20) and both classes were recalled at 94%, indicating a favourable bias–variance balance.

However, Placing one RBF centre at every training point can scale poorly both the feature matrix Φ and the gradient computation become expensive as N grows. A principled remedy would be to select a smaller set of representative centres, for example via k-means clustering or a k-nearest-neighbour (k-NN) approach, retaining only the k cluster centroids or the k most informative training points as centres. This would reduce the feature dimension from N to $k \ll N$, lower the condition number of Φ, and likely improve generalisation further by removing redundant, closely-spaced centres. We may also implement a more principled approach would use cross-validation or a validation set to select l automatically, as well as adding an ℓ_2 penalty on w (i.e. a Gaussian prior in the Bayesian view) would provide an additional lever to control overfitting, particularly relevant at $l = 0.1$ where the ill-conditioned feature matrix amplifies weight magnitude. Finally, all results are sensitive to the random train–test split; reporting averages over multiple random seeds would give a more reliable estimate of generalisation performance. This exercise also demonstrates why we have moved away from RBF's despite them being a universal approximator. Methods like deep neural networks learn hierarchical feature representations that scale efficiently to large datasets, while kernel methods like SVMs achieve sparsity by selecting only support vectors. The RBF approach examined here occupies a middle ground: it demonstrates the power of non-linear feature transformations, but its computational cost and sensitivity to hyperparameters make it less practical than either extreme for contemporary applications. Nonetheless, understanding RBF classifiers provides valuable insight into the bias-variance tradeoff and the importance of feature engineering.