3F8: Inference
Short Lab Report

Joseph Johnson

March 12, 2021

Abstract

This short experiment explores the process of designing and building a simple binary classifier. We expect a well trained classifier to be able to assign unseen input data to it's correct class, either through soft or hard decisions. In this case, the input data is two-dimensional, and there are only two classes. Python was used to process the input data and train a classifier model using iterative gradient ascent, then subsequently plot the data to gain further insight. A key finding of this experiment was that training the model using only linear functions of the data did not provide satisfactory results. Instead, a solution was to extend the data with radial basis functions which, when the right length scale was chosen, generated a very accurate model.

1 Introduction

Classification models aim to solve the problem of assigning unseen data to labels. They are, unlike clustering, a form of supervised learning meaning correctly labelled data is provided as part of the training process. Classification problems are ubiquitous in today's signal processing problems, from semantic segmentation of images in cars to spam filtering of emails. We start by studying a logistic regression model for a binary classifier and explore some of the underlying mathematical principles. We then train a basic model using only linear functions of the data and collect some metrics on it's performance. Following this, we extend the data to a radial basis function representation and train a new model for comparison with the old.

2 Exercise a)

In this exercise we have to consider the logistic classification 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 written in index form as

$$\frac{\partial \mathcal{L}(\underline{\beta})}{\partial \underline{\beta}} = \sum_n \underline{x}^{(n)} \big(y^{(n)} - \sigma(\underline{\beta}^T \underline{x}^{(n)})\big)$$

or in vector form as

$$\frac{\partial \mathcal{L}(\underline{\beta})}{\partial \underline{\beta}} = X^T \cdot \big(\underline{y} - \underline{\sigma}(\underline{\beta}^T \underline{X})\big)$$

3 Exercise b)

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

```
Function estimate_parameters:

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

   Code:
      b := random array of length dimension(x)
      iterate:
        get gradient array of length dimension(x)
        b = b + learning rate * gradient
      return b
```

The learning rate parameter η is chosen by trial and error, such that the solution converges in a reasonable number of steps, without oscillating.

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 analysing Figure 1 we conclude that a linear classifier is unlikely to perform well on this dataset as there is no straight line that can separate the two blue clusters from the central red cluster.

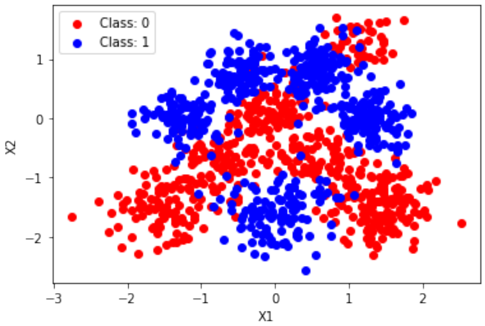


Figure 1: Visualisation of the data.

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 calc_grad(beta, x, y):
  dot_prod = np.matmul(x, beta)
  sigmas = logistic(dot_prod)
  return np.dot(x.T, y - sigmas)

def train_model(x_tilde_train, y_train, x_tilde_test, y_test, alpha, n):
  beta = np.random.normal(size=x_tilde_train.shape[1])
  for i in range(n):
    beta = beta + alpha * calc_grad(beta, x_tilde_train, y_train)
  return beta
```

We then train the classifier using this code. We fixed the learning rate parameter alpha to be $\eta = 10^{-4}$. 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 gradient ascent has converged well and there is not significant over-fitting in the model.

Figure 1 displays the visualisation of the contours of the class predictive probabilities on top of the data. This figure shows that, as expected, the model has tried it's best to draw a decision boundary but it can only do so much with a straight line. When a hard decision is made with threshold probability 0.5, we see that the classifier incorrectly classifies the top red and bottom blue clusters. With some thought we can see why this might be: the model is favouring making correct decisions on the larger clusters.

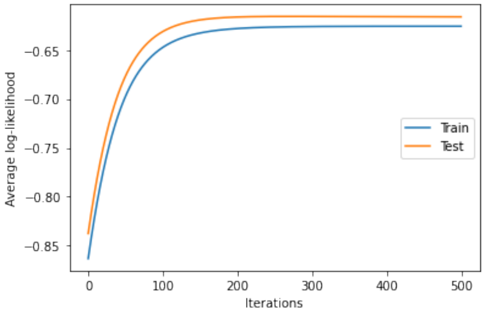


Figure 2: Learning curves showing the average log-likelihood on the training and test datasets.

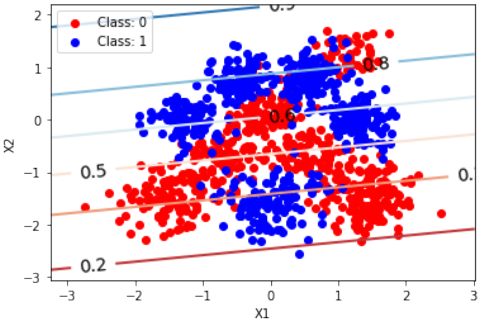


Figure 3: Visualisation of the contours of the class predictive probabilities.

6 Exercise e)

The final average training and test log-likelihoods are shown in Table 1. These results indicate that the model is performing similarly on both datasets, with a modest increase in log-likelihood over the randomly generated beta. The 2x2 confusion matrix is shown in Table 2. The classifier correctly labelled about 66% of the data and was about equally split in its errors across both classes.

Avg. Train ll	**Avg. Test ll**
-0.6256	-0.6160

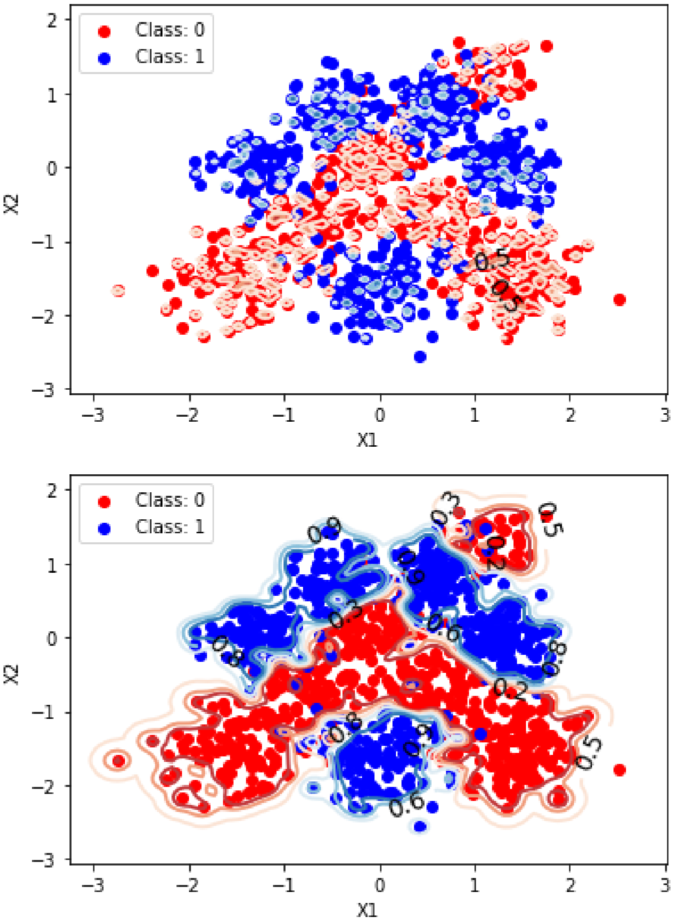
Table 1: Average training and test log-likelihoods.

		$\hat{y}$	
		0	1
y	0	0.375	0.165
	1	0.100	0.360

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 $\eta = \{0.01, 0.001, 0.0001\}$ for each $l = \{0.01, 0.1, 1\}$, respectively. Figure 4 displays the visualisation of the contours of the resulting class predictive probabilities on top of the data for each choice of $l = \{0.01, 0.1, 1\}$.



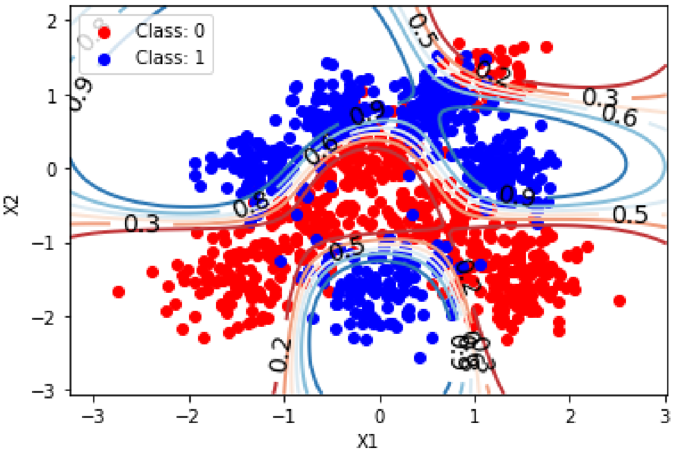


Figure 4: Visualisation of the contours of the class predictive probabilities for $l = 0.01$ (top), $l = 0.1$ (middle), $l = 1$ (bottom).

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 all the models performed well on the training data, however the largest l provided the best performance on test data as expected by its larger, more general contours. 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 the smallest l performed very poorly, labelling 90% of the data as 1, which is clearly not correct. The other two models were very accurate, correctly labelling 90% of the data. When we compare these results to those obtained using the original inputs we conclude that the use of radial basis function has significantly improved the performance of the model, making classification on this dataset viable. The model training time increased negligible, and so we have achieved a drastic increase in performance for little extra cost.

Avg. Train ll	**Avg. Test ll**
-0.02131	-0.6569

Table 3: Results for $l = 0.01$

Avg. Train ll	**Avg. Test ll**
-0.1185	-0.2527

Table 4: Results for $l = 0.1$

Avg. Train ll	**Avg. Test ll**
-0.2219	-0.1857

Table 5: Results for $l = 1$

		$\hat{y}$	
		0	1
y	0	0.530	0.010
	1	0.360	0.100

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

		$\hat{y}$	
		0	1
y	0	0.480	0.06
	1	0.030	0.430

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

		$\hat{y}$	
		0	1
y	0	0.490	0.005
	1	0.010	0.450

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

https://www.overleaf.com/project/603aadea4e706d697a7fa07a https://www.overleaf.com/project/603aadea4e706d697a7fa

9 Conclusions

In this report we demonstrated an implementation of a binary classifier. We first saw this using logistic regression with linear functions of the data as inputs. We noted that the performance of this model was quite poor, and we would not likely use this model in a practical scenario. We then extended the model to include non-linear functions of the data as input and found that the model performs very well when the right length scale is chosen. Additionally, the use of these radial basis functions did not significantly increase the time required to train the model. We could feasibly see how the best performing radial basis function models might be deployed in a real scenario. In terms of limitations, the model was only trained on 800 datapoints, and so we might like to gather more data if we want to use the model to inform or make decisions.