Machine Learning: Assignment 1, Part 2

Pedro Pereira Sarmento

Queen Mary, University of London

November 15, 2019

1 Logistic Regression

Below is presented an implementation of the sigmoid function, the hypothesis in the logistic regression approach, plotted in Figure 1.

```
def sigmoid(z):
    output = 0.0
    output = 1 / (1 + np.exp(-z))
    return output
```

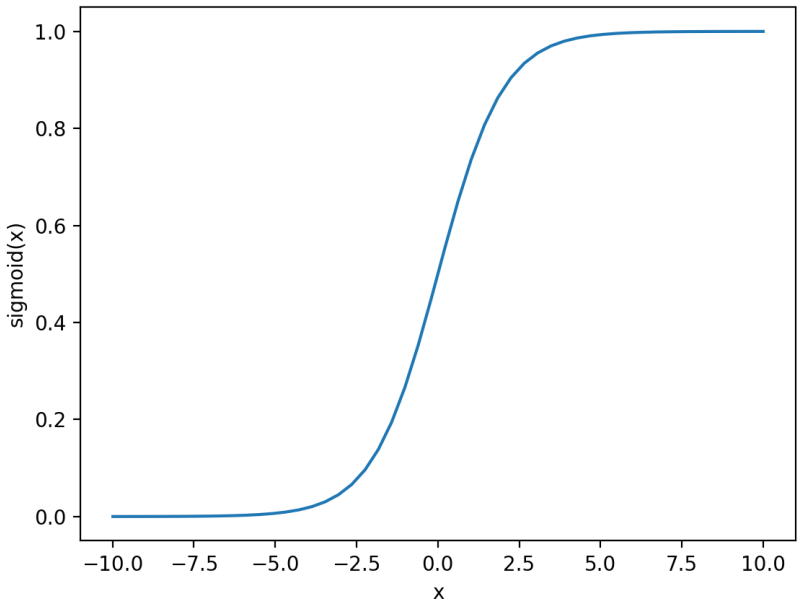

Fig. 1: Plot of the sigmoid function.

In Figures 2 and 3 the plots of the data with or without normalisation are presented.

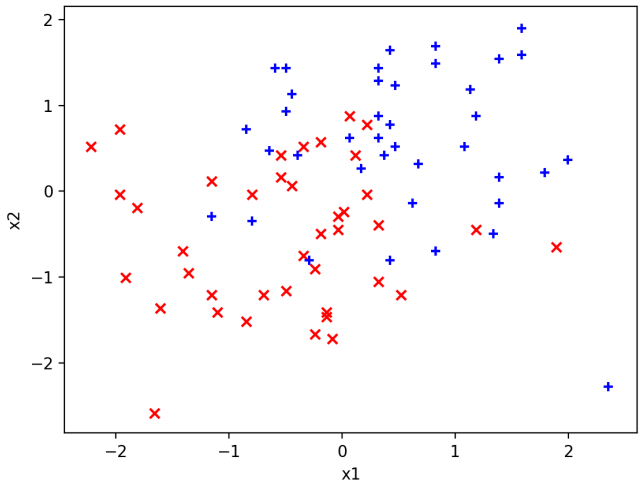

Fig. 2: Plot of the data with normalisation.

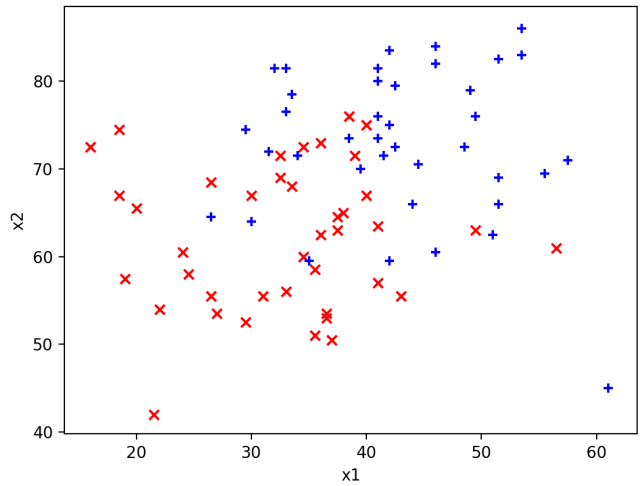

Fig. 3: Plot of the data without normalisation.

1.1 Cost Function and Gradient for Logistic Regression

The modified code for the cost function is displayed below.

```
def compute_cost(X, y, theta):
    J = 0.0
    m = y.shape[0]
    for i in range(m):
        hypothesis = calculate_hypothesis(X, theta, i)
        output = y[i]
        cost = (-1) * output * np.log(hypothesis) -
                (1 - output) * np.log(1 - hypothesis)
        J += cost
    J = J/m
    return J
```

After some tuning of α (learning rate), the resulting cost of $J(\theta) = 0.40545$ was achieved, its plot presented in Figure 4.

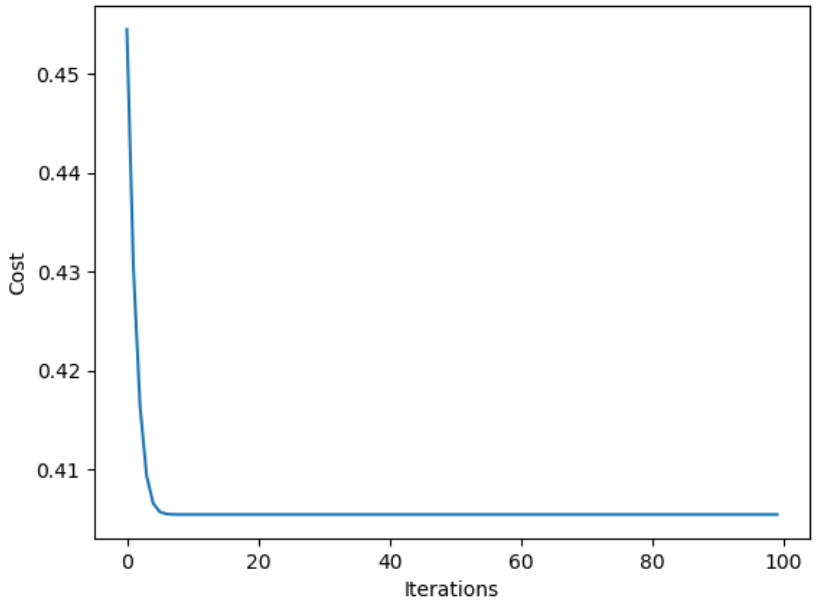

Fig. 4: Plot of the cost for the logistic regression ($J(\theta) = 0.40545$).

1.2 Decision Boundary

In Figure 5 is presented the plot of the decision boundary.

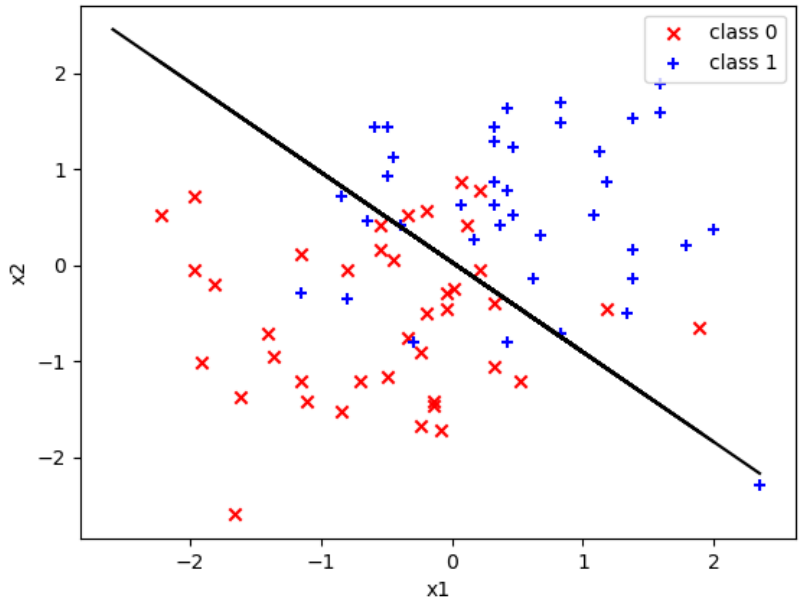

Fig. 5: Plot of the decision boundary.

1.3 Non-linear Features and Overfitting

Under these splitting conditions, is hard to achieve good generalisation. For instance, when the training error is less than 0.1, the test error often exceeds 0.7, as shown in Figures 6 and 7 as an example of bad generalisation. This is the case when the randomly selected 20 points of the data set are distinct but closely clustered within both classes, facilitating a classification on the training set due to its characteristics. Here a good generalisation of the model becomes hard to achieve since its training was performed under very specific conditions, not encompassing examples of a more broad, real, chaotic situation (the case in which clusters in the training set aren't so explicit).
In contrast, a moderately good generalisation scenario is showed in Figures 8 and 9. An analogy can be done here with what was exposed for the bad generalisation example. Because training was performed with regard to a more scattered training data set, the model was able to generalise better.

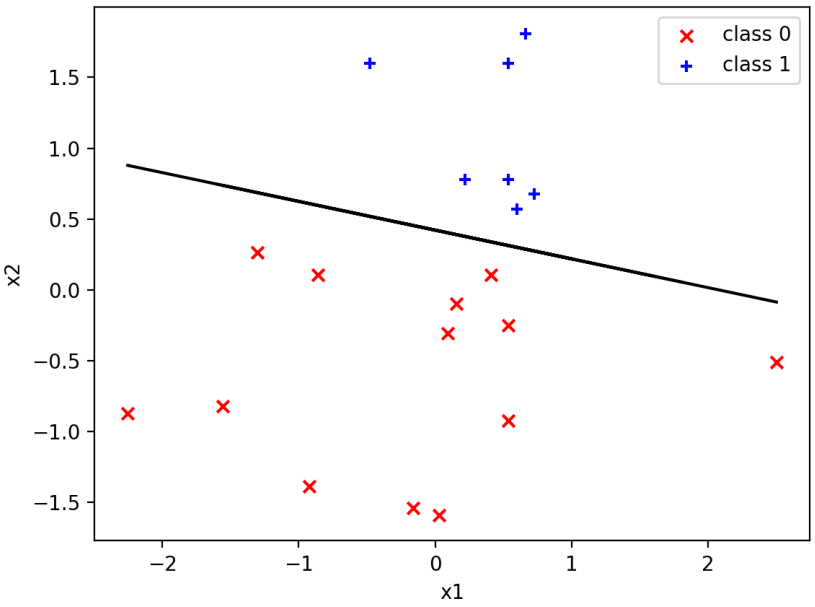

Fig. 6: Training data with a final cost $J(\theta) = 0.06594$.

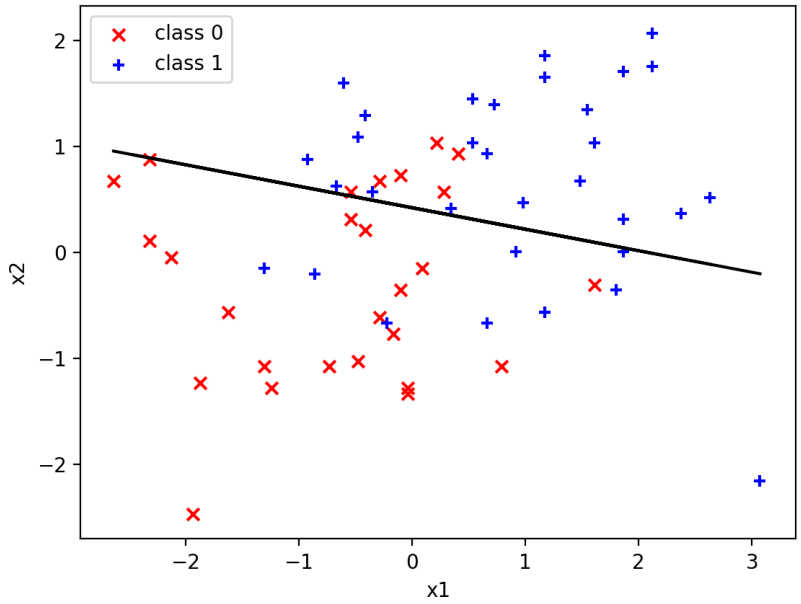

Fig. 7: Test data with a final cost $J(\theta) = 0.94054$.

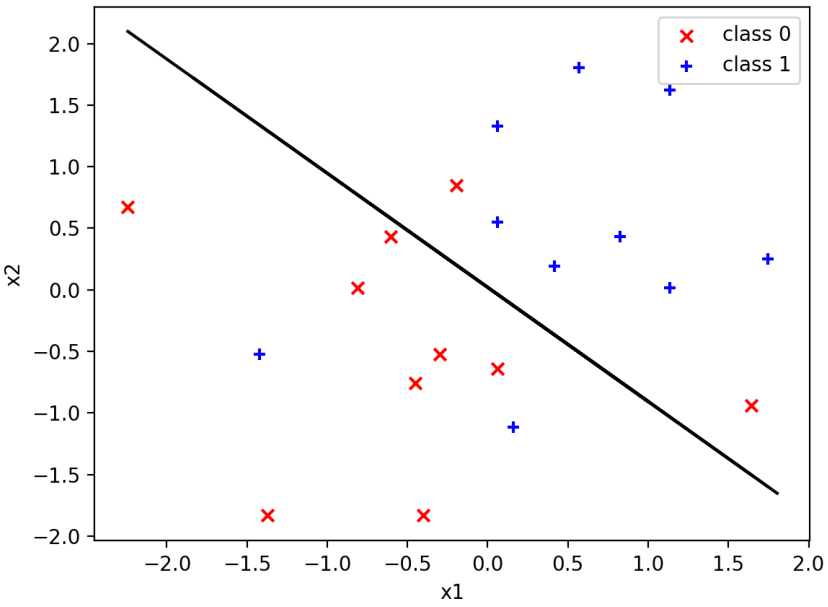

Fig. 8: Training data with a final cost $J(\theta) = 0.43007$.

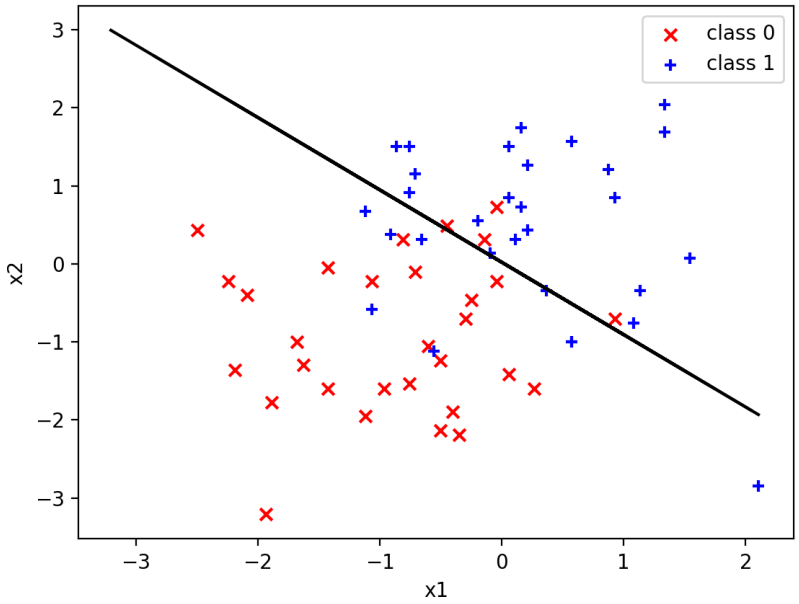

Fig. 9: Test data with a final cost $J(\theta) = 0.43384$.

Furthermore, extending to a 5D, non-linear feature vector, is an attempt to describe the data better with more features.

```
X1sq = np.array([X[:,0] ** 2]).reshape(X.shape[0],1)
X2sq = np.array([X[:,1] ** 2]).reshape(X.shape[0],1)
X1X2 = np.array([X[:,0] * X[:,1]]).reshape(X.shape[0],1)
X = np.concatenate([X, X1X2, X1sq, X2sq], axis=1)
X_normalized, mean_vec, std_vec = normalize_features(X)
X_normalized = np.append(column_of_ones, X_normalized, axis=1)
```

The results, presented in Figure 10 show that non-linear features do not contribute much to the logistic regression model, suggesting an attempt to overfitting (previously we obtained a cost of $J(\theta) = 0.40545$ and with the inclusion of non-linear features the cost is $J(\theta) = 0.40261$).

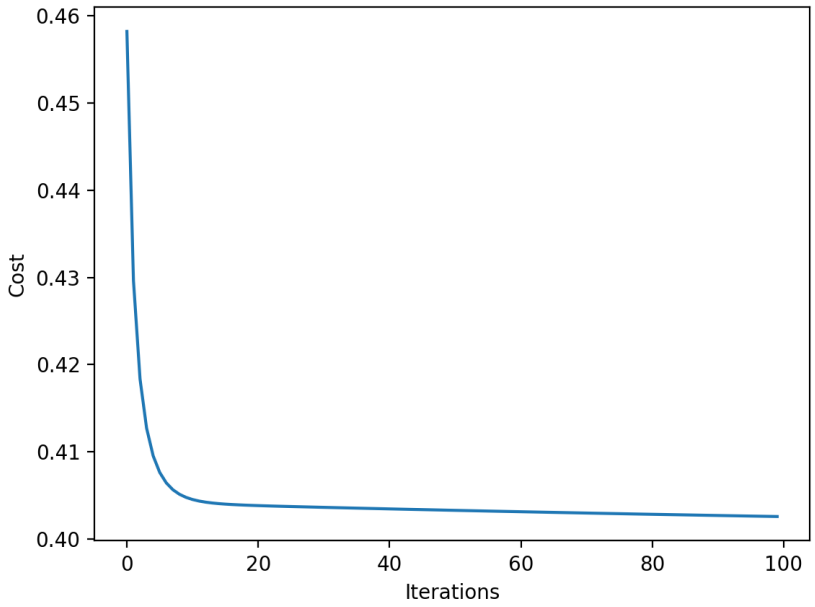

Fig. 10: Cost for 5D non-linear vectorisation of feature space vector $J(\theta) = 0.40261$.

In Task 8, experiments with different sizes of training and test set are proposed. It was found that the best results regarding the test cost occur when the

training set is of size 60 or 70. Below are presented plots of the cost for all the combinations explored.

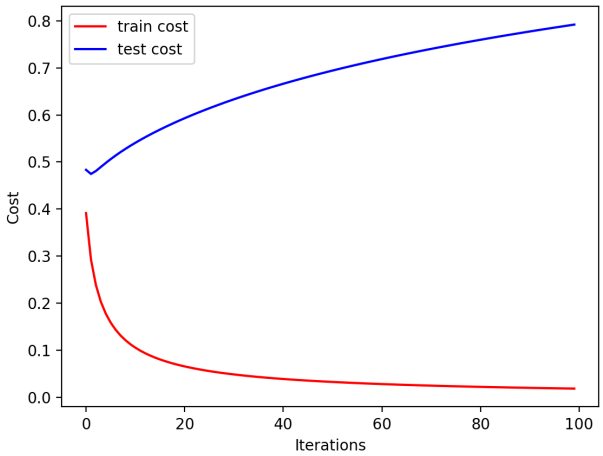

Fig. 11: Cost for a training set size 10, $J_{training}(\theta) = 0.01881$ and $J_{test}(\theta) = 0.79184$.

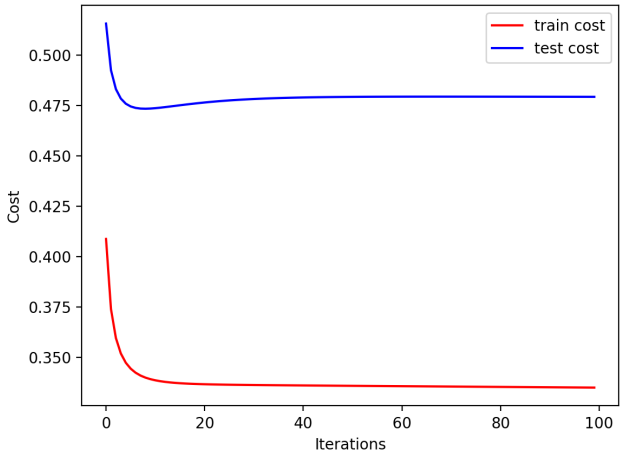

Fig. 12: Cost for a training set size 40, $J_{training}(\theta) = 0.33487$ and $J_{test}(\theta) = 0.47329$.

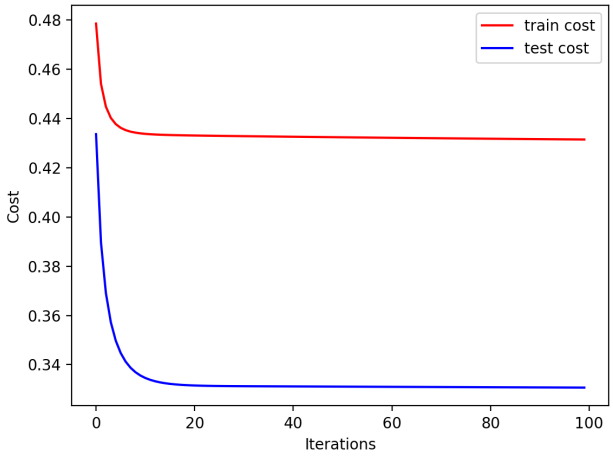

Fig. 13: Cost for a training set size 60, $J_{training}(\theta) = 0.43149$ and $J_{test}(\theta) = 0.33068$.

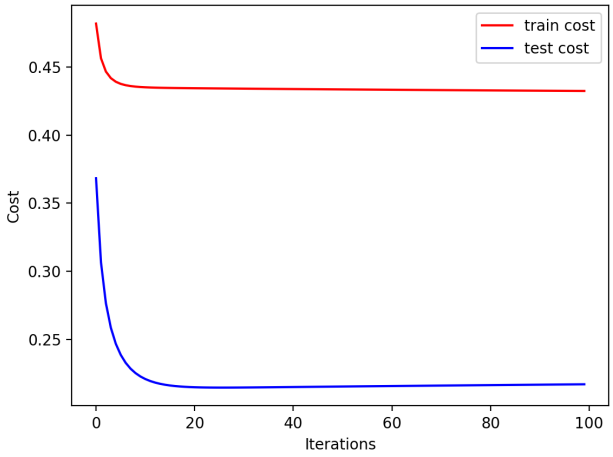

Fig. 14: Cost for a training set size 70, $J_{training}(\theta) = 0.43234$ and $J_{test}(\theta) = 0.21466$.

Findings suggest that increasing the size of the training data, such that it is a larger proportion of the total data, produces a better generalisation than a training data size of 10, since the test cost results to be less than the training cost.

Accordingly, the results of the cost function of a further attempt to include more features, both second-order and third-order polynomials, are presented in Figure 15. The code that produces the inclusion of the mentioned figures is displayed below.

```
X1X2 = np.array([X[:,0] * X[:,1]]).reshape(X.shape[0],1)
X1sq = np.array([X[:,0] ** 2]).reshape(X.shape[0],1)
X2sq = np.array([X[:,1] ** 2]).reshape(X.shape[0],1)
X1cb = np.array([X[:,1] ** 3]).reshape(X.shape[0],1)
X2cb = np.array([X[:,1] ** 3]).reshape(X.shape[0],1)
X = np.concatenate([X, X1X2, X1sq, X2sq, X1cb, X2,cb], axis=1)
X_normalized, mean_vec, std_vec = normalize_features(X)
X_normalized = np.append(column_of_ones, X_normalized, axis=1)
```

By observing Figure 15 we can see that the training cost (red) goes down, but the test cost increases (blue), suggesting that this approach leads the model into overfitting and, consequently, a bad performance in the test data, not being able to achieve a good generalisation.

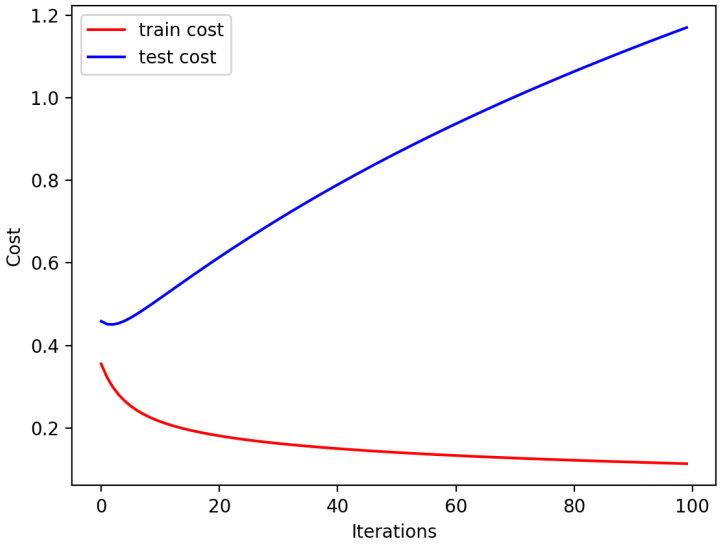

Fig. 15: Cost for 7D non-linear vectorisation of feature space vector $J_{training}(\theta) = 0.11373$ and $J_{test}(\theta) = 0.45074$.

1.4 Classification and XOR

In the case of logistic regression and binary classification, all data points on one side of the classification line are assigned the class of 0, and conversely all others are classified as 1. Despite the classification of XOR, or 'exclusive OR', appears to be very simple problem, Minsky and Papert demonstrated in 1969 that this was in fact a hard problem (concerning the perceptron, the neural network architecture of the time). In Figure 16 is presented an illustration of such finding.

x_1	x_2	y_2
0	0	0
1	0	1
0	1	1
1	1	0

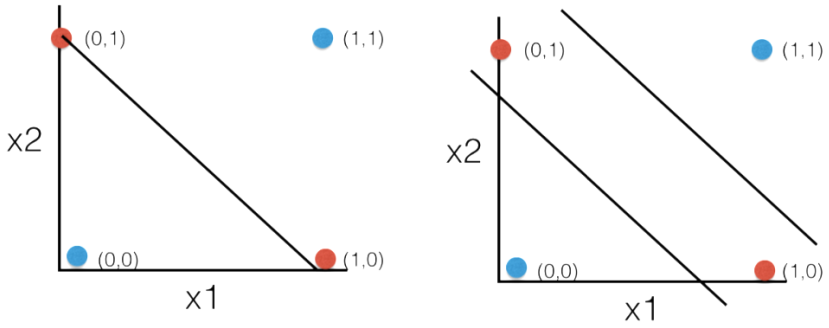

Fig. 16: Graphical representation of XOR classification problem.

Here it is visible that XOR inputs are not linearly separable, as presented on the left hand side of Figure 16. In order to perform this classification, something like what is presented on the right hand side of Figure 16 must be achieved, using 'two lines' instead of one. Concluding, logistic regression cannot solve the XOR classification problem because it only uses 'one line' for class separation.

2 Neural Network

2.1 Implementing Backpropagation on XOR

For the implementation of backpropagation on XOR it was found that a learning rate $\eta = 3$ works well, avoiding overshooting or getting trapped in local optima, outputting results very close to our target values.

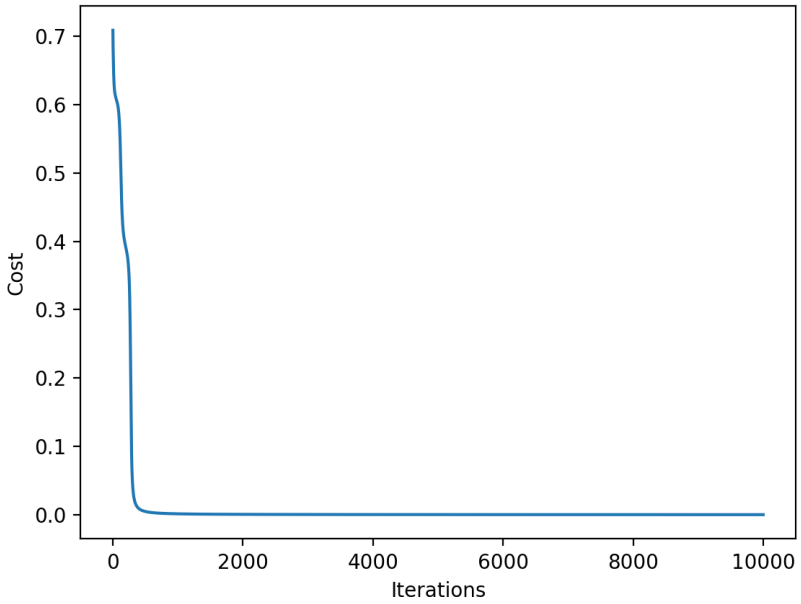

Fig. 17: Error function for a learning rate of 3, with a minimum value of 0.00009.

Furthermore, it was also discovered that for values of the learning rate between 5 and 6, backpropagation got stuck in local optima (error values of 0.40336 and 0.41287, respectively). The predictions are presented in below.

	$\eta = 5$	$\eta = 6$
Target	Prediction	
0	0.00588	0.00475
1	0.64541	0.64553
1	0.64534	0.64549
0	0.64552	0.64559

For Task 11 the logical function AND was chosen. A plot of the error for a successful trial is displayed in Figure 18, using a learning rate of 3 and considering 10000 iterations. As we can see, the cost rapidly decreases and gets very close to zero.

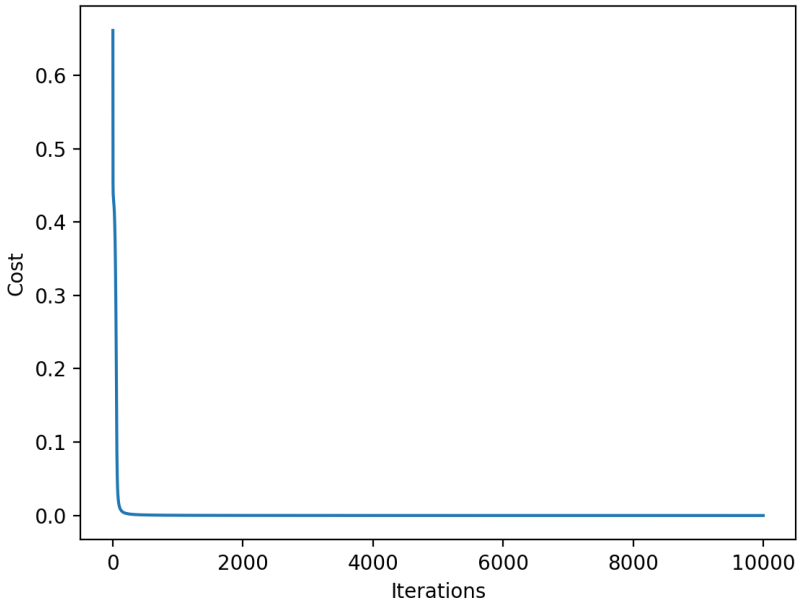

Fig. 18: Error function for a learning rate of 3, with a minimum value of 0.00003.

The obtained prediction values are showed in the table below.

	$\eta = 3$
Target	Prediction
0	0.00017
0	0.00864
0	0.00872
1	0.98990

2.2 Implementing Backpropagation on Iris

Using the model created for the XOR classification problem, here the Iris dataset is utilized, comprising three distinct categories:*Iris setosa*, *Iris virginica* and *Iris versicolor*.
If we were to approach this problem using logistic regression, a 1 vs all heuristic should be followed. This could be accomplished by performing a binary classification for each class against all the rest. In this scenario we would have a classifier $h_i(x)$ for each i class, each trained to classify the class against the remaining classes and finally picking the class that maximises the hypothesis function.
A different approach is followed when considering a neural network in which the hypothesis produces an output vector corresponding to the classifications. This final output will be determined by the sigmoid function and the correspondent values of the hidden layer neurons and weights (obviously, the weights between the input and hidden layer are also considered in the process). Within this architecture, an input could be correctly classified in the output of the neural network.
An illustration of the above given explanation is presented in Figure 19.

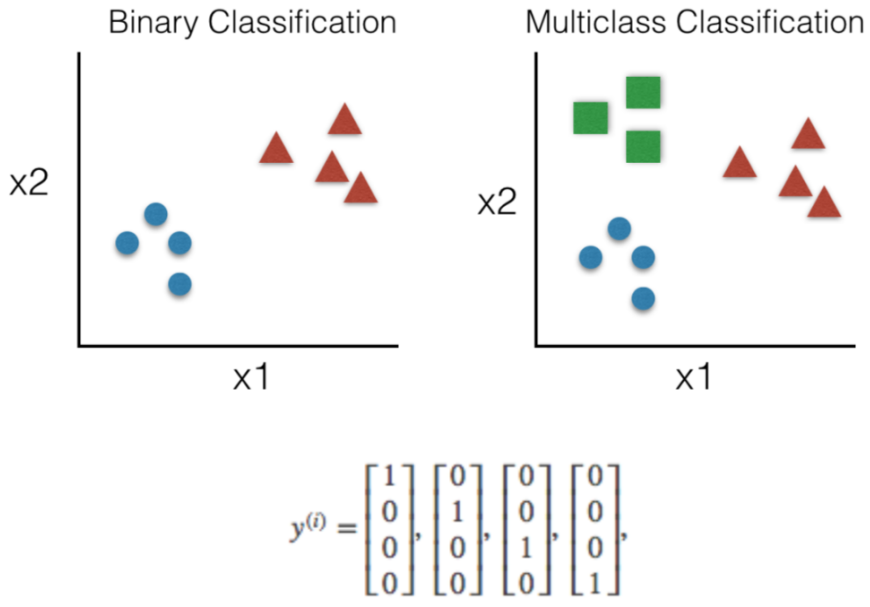

Fig. 19: Logistic Regression vs Neural Network for multi-class classification problem.

Following the neural network approach for the classification of the Iris dataset, in this section we plot the error function for a varying number of neurons in the hidden layer.
For this step, a split of 50-50 between training and test set was performed, each of the sets containing 75 samples. Furthermore, for this problem, a learning rate $\eta = 1$ and a number of 500 iterations were chosen.
The results present in Figures 20 to 25 show that the best result was obtained for a configuration of five neurons in the hidden layer (Figure 20, with $J_{training}(\theta) = 0.00008$ and $J_{test}(\theta) = 0.02204$). Under these conditions, a good generalisation was able to be achieved.

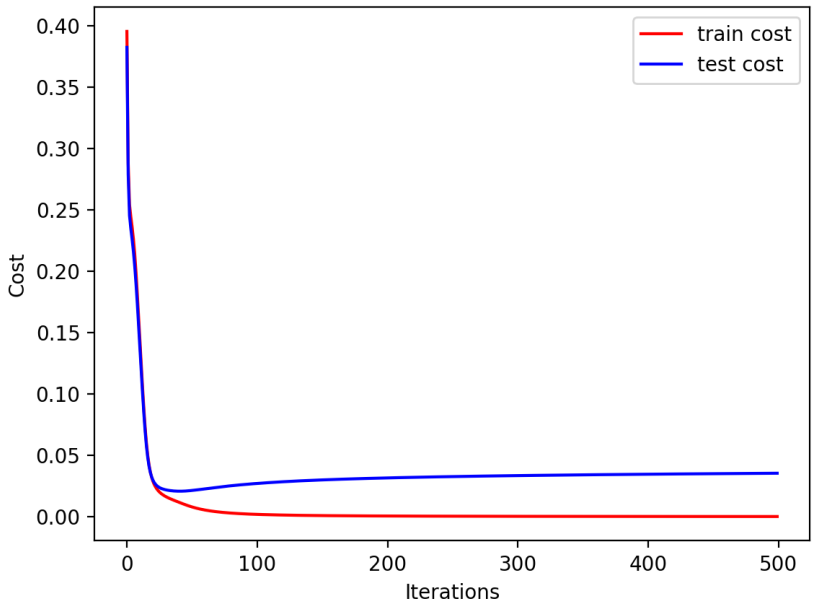

Fig. 20: Cost for five hidden neurons: $J_{training}(\theta) = 0.00008$ and $J_{test}(\theta) = 0.02204$.

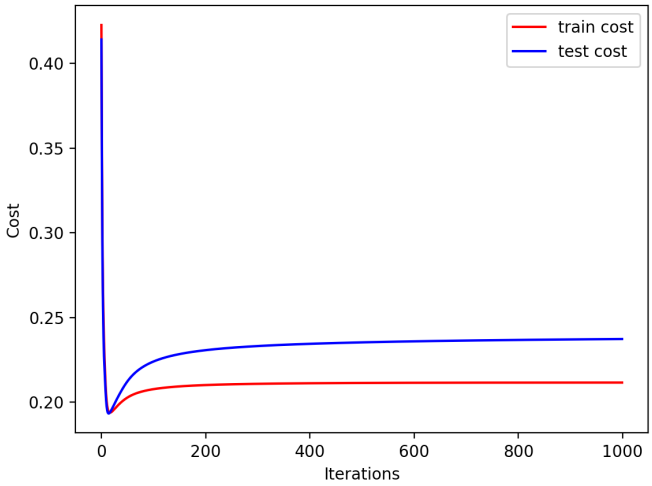

Fig. 21: Cost for one hidden neuron: $J_{training}(\theta) = 0.14535$ and $J_{test}(\theta) = 0.14157$.

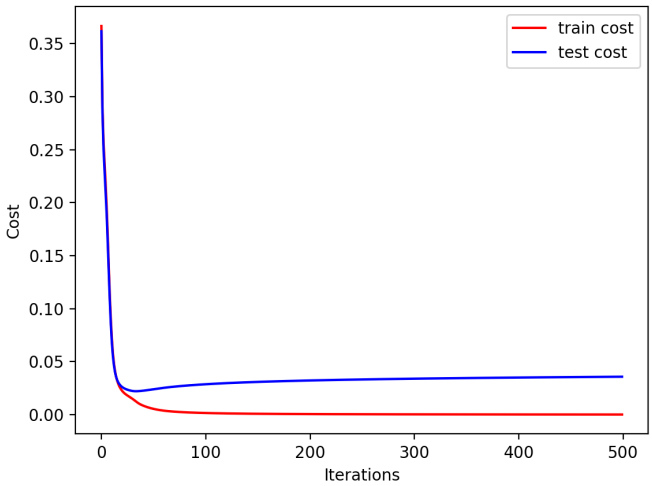

Fig. 22: Cost for two hidden neurons: $J_{training}(\theta) = 0.00022$ and $J_{test}(\theta) = 0.02184$.

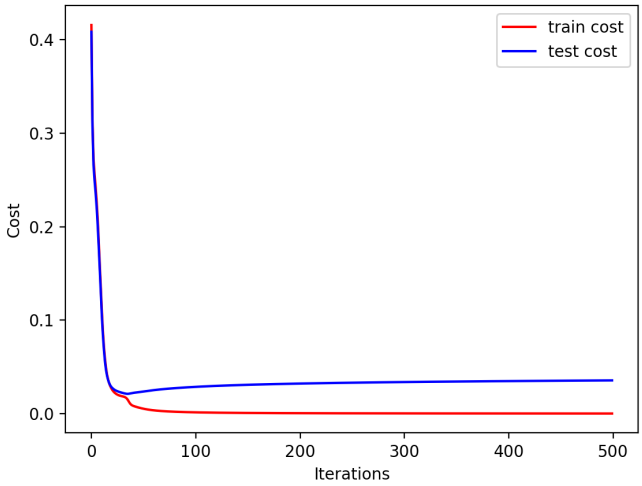

Fig. 23: Cost for three hidden neurons: $J_{training}(\theta) = 0.00018$ and $J_{test}(\theta) = 0.02126$.

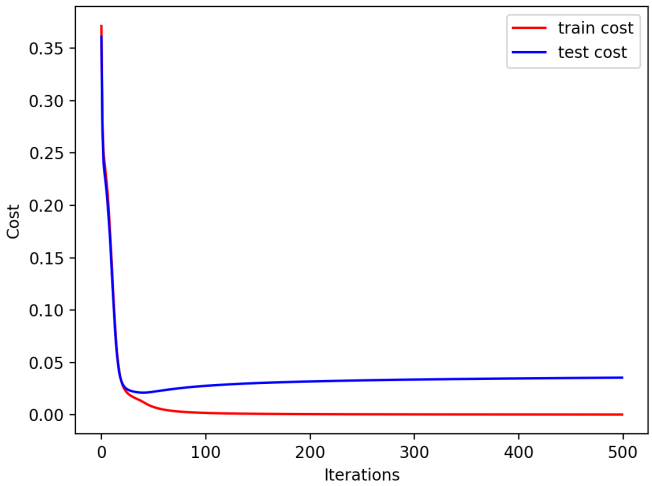

Fig. 24: Cost for seven hidden neurons: $J_{training}(\theta) = 0.00016$ and $J_{test}(\theta) = 0.02105$.

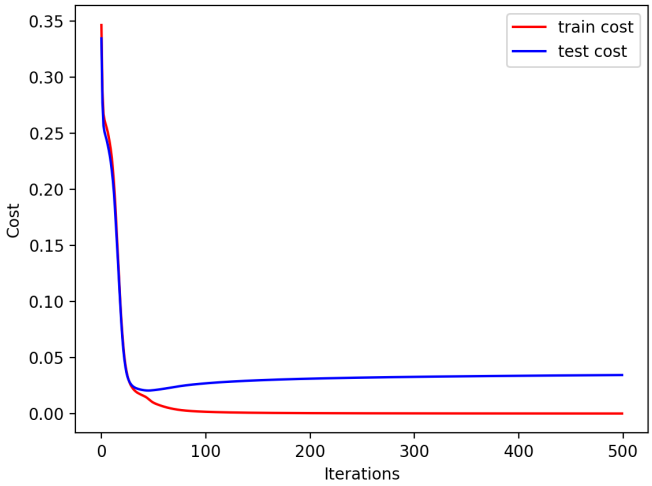

Fig. 25: Cost for ten hidden neurons: $J_{training}(\theta) = 0.00014$ and $J_{test}(\theta) = 0.02063$.