Logistic Classification

3F8 laboratory experiment
Theo Brown
Selwyn College, University of Cambridge

February 24, 2022

Abstract

Enter a short summary here.

1 Introduction

Classification is the task of grouping data points according to their shared features, and assigning each group a label. For a human with a small dataset, this is a trivial task, but it is much more difficult and slow with very large or complex data sets. Consequently, the development of good machine classifiers is an important area in statistical learning. Trained classifiers can be used as predictive tools to identify the most probable class that a new input will belong to, which could, for example, be useful in a medical setting, such as identifying the most at-risk patients.

A key tool in classification is the ability to classify non-linear datasets, where the boundary between classes is not a straight line. This report demonstrates the failure of a linear classifier on a non-linear dataset, and explores the use of a set of non-linear basis functions (in this case, radial functions) applied to the inputs before classifying to allow the discovery of non-linear decision boundaries.

2 Derivation of the gradient of the log-likelihood

Define the n-th input vector as $\mathbf{x}_n$, and let the augmented input vector $\tilde{\mathbf{x}}_n = \left[1, \mathbf{x}_n^T\right]^T$. The set of augmented input vectors are assembled as the columns of matrix $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_N]$. Define $\sigma(x) = \frac{1}{1+e^{-x}}$. The class labels, $\mathbf{y}$, are modelled as being independent and identically generated from a Bernoulli distribution, with $p(y_n = 1|\tilde{\mathbf{x}}_n) = \sigma(\mathbf{w}^T\tilde{\mathbf{x}}_n)$ and $p(y_n = 0|\tilde{\mathbf{x}}_n) = 1 - \sigma(\mathbf{w}^T\tilde{\mathbf{x}}_n) = \sigma(-\mathbf{w}^T\tilde{\mathbf{x}}_n)$, where $\mathbf{w}$ is a vector of model weights. The log-likelihood of $\mathbf{w}$ is therefore:

$$\begin{aligned}\mathcal{L}(w) = \log p(y|\tilde{\mathbf{X}}, \mathbf{w}) &= \log \prod_{n=1}^{N} \sigma(\mathbf{w}^T\tilde{\mathbf{x}}_n)^{y_n}\sigma(-\mathbf{w}^T\tilde{\mathbf{x}}_n)^{1-y_n} \\ &= \sum_{n=1}^{N} y_n \log \sigma(\mathbf{w}^T\tilde{\mathbf{x}}_n) + (1-y_n)\log\sigma(-w^T\tilde{\mathbf{x}}_n)\end{aligned}$$

The derivative of the log-likelihood is calculated using the identities $\frac{d\sigma(x)}{dx} = \sigma(x)\sigma(-x)$ and $\frac{d\log(x)}{dx} = \frac{1}{x}$. Applying the chain rule:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial w} &= \sum_{n=1}^{N} y_n\sigma(-\mathbf{w}^T\tilde{\mathbf{x}}_n)\tilde{\mathbf{x}}_n - (1-y_n)\sigma(\mathbf{w}^T\tilde{\mathbf{x}}_n)\tilde{\mathbf{x}}_n \\ &= \sum_{n=1}^{N} \left(y_n - \sigma(\mathbf{w}^T\tilde{\mathbf{x}}_n)\right)\tilde{\mathbf{x}}_n \\ &= \tilde{\mathbf{X}}(\mathbf{y} - \sigma(\tilde{\mathbf{X}}^T\mathbf{w}))\end{aligned}$$

3 Gradient ascent algorithm

To find the parameter $\mathbf{w}$ that maximises the log-likelihood, a gradient ascent algorithm is used. Define $\tilde{\mathbf{X}}$ and y as in Section 2, then let n be the number of iterations to run the algorithm for and r be the learning rate. The algorithm is defined as follows:

procedure GradientAscent($\tilde{\mathbf{X}}$, $\mathbf{y}$, n, r)
  $\mathbf{w} \leftarrow [0 \dots 0]^T$ ▷ Initialise the weights
  for $i \leftarrow 1, n$ **do**
    $\mathbf{d} \leftarrow \tilde{\mathbf{X}}(\mathbf{y} - \sigma(\tilde{\mathbf{X}}^T \mathbf{w}))$ ▷ Calculate the gradient
    $\mathbf{w} \leftarrow \mathbf{w} + r\mathbf{d}$ ▷ Move along the gradient, scaled by learning rate
  end for
end procedure

Note that the weights do not necessarily need to be initialised as zeros - they could have been selected randomly, or initialised as ones, for example. Provided the number of iterations is large enough and the algorithm does not get stuck in a local maximum, the effect of the initial values should be negligible.

The learning rate, n, is chosen empirically. Starting with a value chosen such the algorithm shows signs of convergence, the learning rate is increased until the algorithm begins to oscillate. The rate is then decreased slightly so that the oscillation is avoided. Using this method, a fast rate of convergence is ensured and divergence avoided.

In Python[1], the gradient ascent algorithm can be implemented as follows, noting the necessary alterations from column to row vectors:

```
import numpy as np

def logistic(x):
    return 1 / (1 + np.exp(-x))

def gradient_ascent(X, y,
                    number_iterations, learning_rate):
    # Prepend a column of ones to the input data (X)
    X1 = np.column_stack((np.ones(X.shape[0]), X))
    # Initialise the weights
    w = np.ones(X1.shape[1])

    for i in range(number_iterations):
        # Calculate the gradient
        dL = (y - logistic(X1 @ w)) @ X1
        # Move in the direction of the gradient,
        # scaled by learning rate
        w += learning_rate * dL
    return w
```

4 Data visualisation

The data is visualised in the two-dimensional input space in Figure **??**. From the plot it is clear that a linear classifier will perform badly on the data, as the classes are not linearly separable: there is no straight boundary that successfully distinguishes between red and blue data points.

5 Linear classifier training

The data was split into a training set of 800 points and a test set of 200 points, and the gradient descent algorithm with a learning rate of 0.001 was applied to find the weights for the linear classifier. The mean log-likelihood (where the mean is taken across all of the weights) is plotted in Figure **??** for the training and test datasets at each iteration.

Figure **??** shows that a plateau is reached for the log-likelihood of the parameters in both the training and test sets, after which no further improvements can be made. This suggests an optimum has been

[1]The Python 3.5+ matrix multiplication operator @ provides an easy method of vectorising the operations

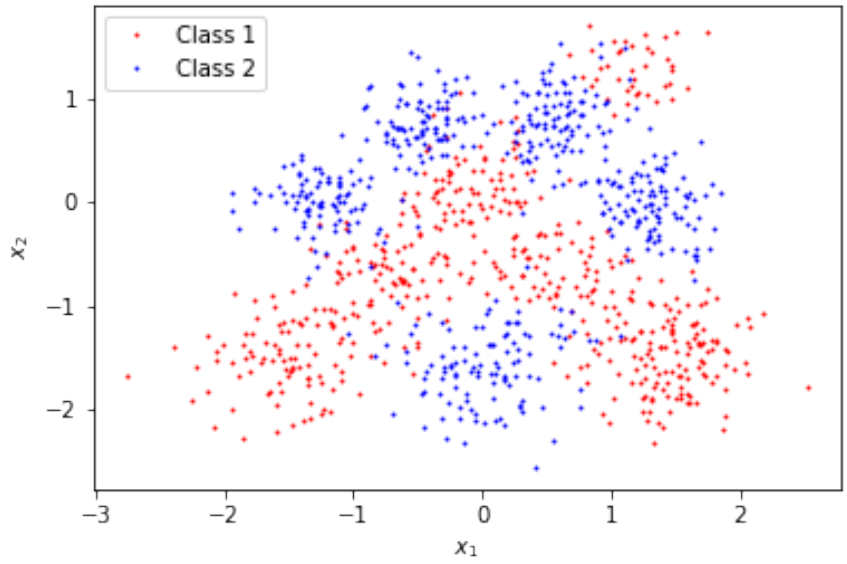

Figure 1: Plot of the two-dimensional input features (1000 datapoints, 2 classes)

found for this model. After training there remains a difference between the performance on the test and training sets, but not at a significant level (see Section 6).

6 Linear classifier performance

The predictive distribution of the model is shown in Figure **??**. It is clear that this model fails to accurately classify the two classes. It identifies that the general principle that more of the points with $x_2 < 0$ are class 1 and more of the points with $x_2 > 0$ values are class 2, but it is apparent to the human eye that this is overly simplified and fails to identify the 'islands' where one class appears surrounded by the other.

The mean log-likelihood of the trained model evaluated on the training and test sets is shown in Figure **??**. It shows a difference of 3-4% between the training and test data, which is sufficiently small to suggest that overfitting has been avoided. As expected, the performance is marginally better for the training set, as this was the data that the model was fit to.

To find the predicted class label for a point $\boldsymbol{x}_n$, a hard threshold is applied to the predicted class probabilities:

$$\hat{y}_n = \begin{cases} 0 & \sigma(\tilde{\boldsymbol{x}}_n^T \boldsymbol{w}) \leq 0.5 \\ 1 & \sigma(\tilde{\boldsymbol{x}}_n^T \boldsymbol{w}) > 0.5 \end{cases}$$

Using the predicted class labels, the confusion matrix for the test data is found (Figure **??**). The confusion matrix is a good method of assessing the classifier's performance, as it shows the fraction of correct and incorrect classifications for each class.

Figure **??** shows that the linear classifier successfully identifies class 0 about 75% of the time. Comparing with Figure **??**, we can see that the decision boundary (shown as a contour of value 0.0) correctly separates roughly four out of six visually identifiable clusters of class 0, and fails to account for two. Similarly, the classifier identifies class 1 70% of the time, and the decision boundary separates approximately four out of five visually identifiable class 1 clusters.

We conclude that the linear classifier does not provide a particularly useful result for this dataset. A new method is presented in the next section, which uses a non-linear basis function to allow the classifier to identify non-linear class boundaries.

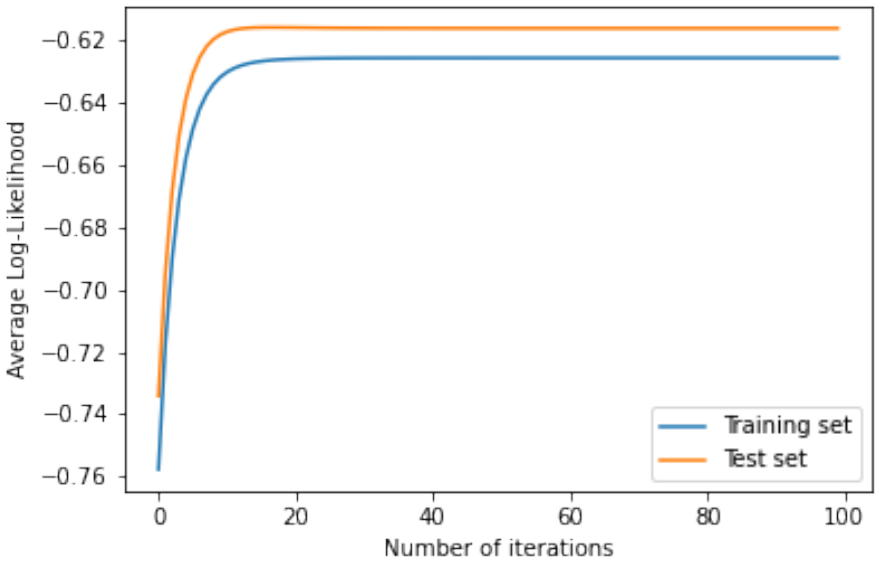

Figure 2: Plot of the mean log-likelihood of the weights for the test and training data, over the course of model training

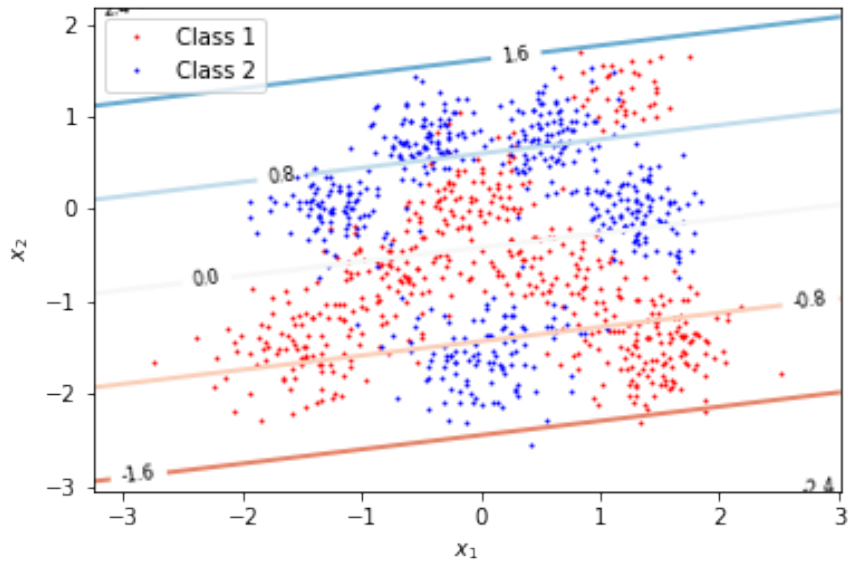

Figure 3: Model predictive distribution overlaid on input datapoints coloured by class

Performance metric	**Value**
Mean log-likelihood, training data	-0.629
Mean log-likelihood, test data	-0.604
$\begin{pmatrix} P(\hat{y}=0\|y=0) & P(\hat{y}=1\|y=0) \\ P(\hat{y}=0\|y=1) & P(\hat{y}=1\|y=1) \end{pmatrix}$	$\begin{pmatrix} 0.764 & 0.235 \\ 0.276 & 0.723 \end{pmatrix}$

Figure 4: Performance metrics for the linear classifier