Learning from Data
Logistic Regression

Course lecturer: Amos Storkey
`a.storkey@ed.ac.uk`
Course page : `http://www.anc.ed.ac.uk/~amos/lfd/`

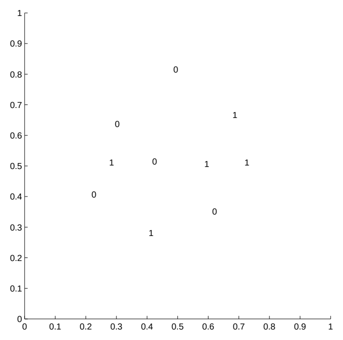

Figure 1:

1 Introduction

A common application of machine learning is to classify a novel instance $\mathbf{x}$ as belonging to a particular class. Here we concentrate on only two class problems. Explicitly, we are given some training data, $D = \{(\mathbf{x}^\mu, t^\mu), \mu = 1 \dots P\}$, where the targets $c \in \{0, 1\}$. An example is given is given in fig(1) in which the training inputs $\mathbf{x}$ are two dimensional real values, and the associated target values are plotted.

We need to make an assignment for a novel point $\mathbf{x}$ to one of the two classes. More generally, we can assign the probability that a novel input $\mathbf{x}$ belongs to class 1

$$p(c = 1|\mathbf{x}) = f(\mathbf{x}; \mathbf{w}) \tag{1.1}$$

where f is some function parameterised by $\mathbf{w}$. Since the function $f(x)$ represents a probability, $f(x)$ must be bounded between 0 and 1.

In previous chapters we have used class conditional density estimation and Bayes rule to form a classifier $p(c|\mathbf{x}) \propto p(\mathbf{x}|c)p(c)$. Here, we take the direct approach and postulate a model explicitly for $p(c|\mathbf{x})$. There are advantages and disadvantages in both of these approaches – my personal favourite is to try the indirect approach more often than the direct approach.

Logistic Sigmoid Function

One of the simplest choices of function is the sigmoid function, $f(x) = 1/(1 + \exp(-x))$, which is plotted in fig(2). What about the argument of

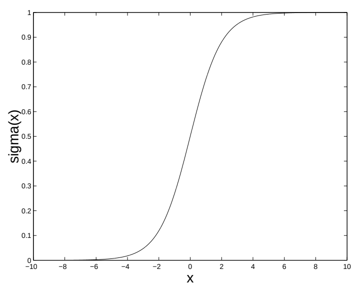

Figure 2: The logistic sigmoid function $\sigma(x) = 1/(1 + e^{-x})$.

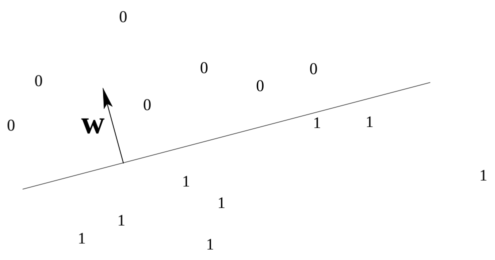

Figure 3: The decision boundary $p(c = 1|\mathbf{x}) = 0.5$ (solid line). For two dimensional data, the decision boundary is a line. If all the training data for class 1 lie on one side of the line, and for class 0 on the other, the data is said to be *linearly separable.*

the function f? Logistic regression corresponds to the choice

$$p(c = 1|\mathbf{x}) = \sigma(b + \mathbf{x}^T\mathbf{w}) \tag{1.2}$$

where b is a constant scalar, and $\mathbf{w}$ is a constant vector. When the argument of the sigmoid function $b+\mathbf{x}^T\mathbf{w}$ is above zero, the probability that the input point $\mathbf{x}$ belongs to class 1 is above 0.5. The greater the argument value is, the higher is the probability that $\mathbf{x}$ is in class 1 (according to our logistic regresssion model). Similarly, the more negative is the argument, the more likely it is that $\mathbf{x}$ belongs to class 0.

Linear (Hyperplane) Decision Boundary

The hyperplane $b + \mathbf{x}^T\mathbf{w} = 0$ forms the decision boundary (where $p(c = 1|\mathbf{x}) = 0.5$) – on the one side, examples are classified as 1's, and on the other, 0's. The "bias" parameter b simply shifts the decision boundary by a constant amount. The orientation of the decision boundary is determined by $\mathbf{w}$ – indeed, $\mathbf{w}$ represents the normal to the hyperplane. To understand this, consider a new point $\mathbf{x}^* = \mathbf{x}+\mathbf{w}^\perp$, where $\mathbf{w}^\perp$ is a vector perpendicular to $\mathbf{w}$ ($\mathbf{w}^T\mathbf{w}^\perp = 0$). Then

$$b + \mathbf{w}^T\mathbf{x}^* = b + \mathbf{w}^T\left(\mathbf{x} + \mathbf{w}^\perp\right) = b + \mathbf{w}^T\mathbf{x} + \mathbf{w}^T\mathbf{w}^\perp = b + \mathbf{w}^T\mathbf{x} = 0 \tag{1.3}$$

Thus if $\mathbf{x}$ is on the decision boundary, so is $\mathbf{x}$ plus any vector perpendicular to $\mathbf{w}$. In n dimensions, the space of vectors that are perpendicular to $\mathbf{w}$ occupy an $n-1$ dimensional linear subspace, in otherwords an $n-1$ dimensional hyperplane. For example, if the data is two dimensional, the decision boundary is a one dimensional hyperplane, a line. This situation is depicted in fig(3). If all the training data for class 1 lie on one side of the line, and for class 0 on the other, the data is said to be *linearly separable.*

Classification confidence

We plot $\sigma(b + \mathbf{x}^T\mathbf{w})$ for different values of $\mathbf{w}$ in fig(4) and fig(5). The decision boundary is at $\sigma(\mathbf{x}) = 0.5$. Note how the classification becomes more confident as the size of the weight vector components increases – that is, as

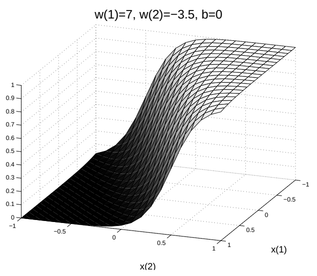

Figure 4: The logistic sigmoid function $\sigma(x) = 1/(1 + e^{-x})$, with $x = \mathbf{w}^T\mathbf{x} + b$.

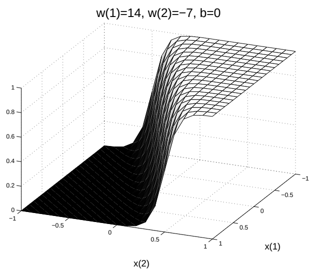

Figure 5: The logistic sigmoid function $\sigma(x) = 1/(1+e^{-x})$, with $x = \mathbf{w}^T\mathbf{x}+b$.

we move only a short distance away from the decision boundary, we predict very confidently the class of $\mathbf{x}$ if the weights are large.

The Perceptron

As we have defined it sofar, $\mathbf{x}$ is assigned to class 1 with some probability. It is not certainly in class 1 unless $p(c = 1|\mathbf{x}) = 1$, which cannot happen unless the weights tend to infinity.

The perceptron is a historical simpler model in which $\mathbf{x}$ is assigned to class 1 with complete certainty if $b + \mathbf{w}^T\mathbf{x} \geq 0$, and to class 0 otherwise. Alternatively, we can define a new rule :

$$p(c = 1|\mathbf{x}) = \theta(b + \mathbf{x}^T\mathbf{w}) \tag{1.4}$$

where the "theta" function is defined as $\theta(x) = 1$ if $x \geq 0$, and $\theta(x) = 0$ if $x < 0$. Since the perceptron is just a special case (the deterministic limit) of logistic regression, we develop here training algorithms for the more general case.

1.1 Training

Given a data set D, how can we adjust/"learn" the weights to obtain a good classification? Probabilistically, if we assume that each data point has been drawn independently from the same distribution that generates the data

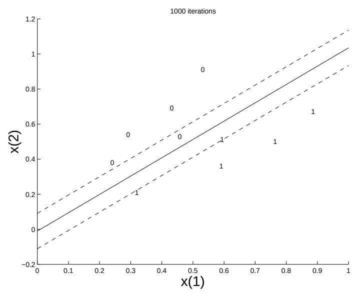

Figure 6: The decision boundary $p(c = 1|\mathbf{x}) = 0.5$ (solid line) and confidence boundaries $p(c = 1|\mathbf{x}) = 0.9$ and $p(c = 1|\mathbf{x}) = 0.1$.

(the standard i.i.d assumption), the likelihood of the observed data is[1]

$$p(D) = \prod_{\mu=1}^{P} p(c^\mu|\mathbf{x}^\mu) = \prod_{\mu=1}^{P} (p(c = 1|\mathbf{x}^\mu))^{c^\mu} (1 - p(c = 1|\mathbf{x}^\mu))^{1-c^\mu} \tag{1.5}$$

Thus the log likelihood is

$$L = \sum_{\mu=1}^{P} c^\mu \log p(c = 1|\mathbf{x}^\mu) + (1 - c^\mu) \log (1 - p(c = 1|\mathbf{x}^\mu)) \tag{1.6}$$

Using our assumed logistic regression model, this becomes

$$L(\mathbf{w}, b) = \sum_{\mu=1}^{P} c^\mu \log \sigma(b + \mathbf{w}^T\mathbf{x}^\mu) + (1 - c^\mu) \log \left(1 - \sigma(b + \mathbf{w}^T\mathbf{x}^\mu)\right) \tag{1.7}$$

1.2 Gradient Ascent

We wish to maximise the likelihood of the observed data. To do this, we can make use of gradient information of the likelihood, and then ascend the likelihood.

The gradient is given by (using $\sigma'(x) = \sigma(x)(1 - \sigma(x))$)

$$\nabla_{\mathbf{w}} L = \sum_{\mu=1}^{P} (c^\mu - \sigma(\mathbf{x}^\mu; \mathbf{w}))\mathbf{x}^\mu \tag{1.8}$$

and the derivative with respect to the biases is

$$\frac{dL}{db} = \sum_{\mu=1}^{P} (c^\mu - \sigma(\mathbf{x}^\mu; \mathbf{w})) \tag{1.9}$$

[1] Note that this is not quite the same strategy that we used in density estimation. There we made, for each class, a model of how $\mathbf{x}$ is distributed. That is, given the class c, make a model of $\mathbf{x}$, $p(\mathbf{x}|c)$. We saw that, using Bayes rule, we can use $p(\mathbf{x}|c)$ to make class predictions $p(c|\mathbf{x})$. Here, however, we assume that, given $\mathbf{x}$, we wish to make a model of the class probability, $p(c|\mathbf{x})$ directly. This does not require us to use Bayes rule to make a class prediction. Which approach is best depends on the problem, but my personal feeling is that density estimation $p(\mathbf{x}|c)$ is worth considering first.

Gradient ascent would then give

$$\mathbf{w}^{new} = \mathbf{w} + \eta \nabla_{\mathbf{w}} L \tag{1.10}$$

$$b^{new} = b + \eta dL/db \tag{1.11}$$

where η, the *learning rate* is a small scalar chosen small enough to ensure convergence of the method (a reasonable guess is to use $\eta = 0.1$). The application of the above rule will lead to a gradual increase in the log likelihood.

Batch version

Writing the above result out in full gives explicitly

$$\mathbf{w}^{new} = \mathbf{w} + \eta \sum_{\mu=1}^{P} (c^\mu - \sigma(\mathbf{x}^\mu; \mathbf{w}))\mathbf{x}^\mu \tag{1.12}$$

$$b^{new} = b + \eta \sum_{\mu=1}^{P} (c^\mu - \sigma(\mathbf{x}^\mu; \mathbf{w})) \tag{1.13}$$

This is called a "batch" update since the parameters $\mathbf{w}$ and b are updated only after passing through the whole (batch) of training data – see the MATLAB code below which implements the batch version (note that this is not written optimally to improve readability). We use a stopping criterion so that if the gradient of the objective function (the log likelihood) becomes quite small, we are close to the optimum (where the gradient will be zero), and we stop updating the weights.

Online version

An alternative that is often preferred to Batch updating, is to update the parameters after each training example has been considered:

$$\mathbf{w}^{new} = \mathbf{w} + \frac{\eta}{P} (c^\mu - \sigma(\mathbf{x}^\mu; \mathbf{w}))\mathbf{x}^\mu \tag{1.14}$$

$$b^{new} = b + \frac{\eta}{P} (c^\mu - \sigma(\mathbf{x}^\mu; \mathbf{w})) \tag{1.15}$$

These rules introduce a natural source of stochastic (random) type behaviour in the updates, and can be useful in avoiding local minima. However, as we shall see below, the error surface for logistic regression is bowl shaped, and hence there are no local minima. However, it is useful to bear in mind the online procedure for other optimisation problems with local minima.

```
% Learning Logistic Linear Regression Using Gradient Ascent (BATCH VERSION)

n0 = 16; x0 = randn(2,n0) + repmat([1 -1]',1,n0); % training data for class 0
n1 = 11; x1 = randn(2,n1) + repmat([-1 1]',1,n1); % training data for class 1

eta = 0.1; % learning rate
w = [0 0]'; b = 0; % initial guess about the parameters
it = 0; itmax = 1000;  % maximum number of iterations
gb = 1; gw = zeros(size(w)); % set gradients initally to ensure at least update

while sum(abs(gw)) + abs(gb) > 0.1  % continue whilst gradient is large

  it = it + 1;  % increment the number of updates carried out
  gb = 0; gw = 0*gw; % reset gradients to zero

  for d = 1:n1 % cycle through the class 1 data
    c = 1 - 1/(1+exp(-(b+w'*x1(:,d))));
    gb = gb + c;
    gw = gw + c*x1(:,d);
  end

  for d = 1:n0 % cycle through the class 0 data
    c = 0 - 1/(1+exp(-(b+w'*x0(:,d))));
    gb = gb + c;
    gw = gw + c*x0(:,d);
  end

  w = w + eta*gw; % update the weight vector
  b = b + eta*gb; % update the bias scalar

  if it > itmax; break; end
end

% calculate the probabilities p(c=1|x) for the training data :
disp('p(c=1|x) for class 1 training data : ');
1./(1+exp(-(repmat(b,1,n1)+w'*x1)))

disp('p(c=1|x) for class 0 training data : ');
1./(1+exp(-(repmat(b,1,n0)+w'*x0)))
```

One important point about the training is that, provided the data is linearly separable, the weights will continue to increase, and the classifications will become extreme. This may be an undesirable situation in case some of the training data has been mislabelled, or a test point needs to be classified – it is rare that we could be absolutely sure that a test point belongs to a particular class. For non-linearly separable data, the predictions will be less certain, as reflected in a broad confidence interval – see fig(7).

The error surface is bowl-shaped

The Hessian of the log likelihood is

$$H_{ij} \equiv \frac{\partial^2 H}{\partial w_i w_j} = -\sum_{i,j,\mu} x_i^\mu x_j^\mu \sigma^\mu (1 - \sigma^\mu) \tag{1.16}$$

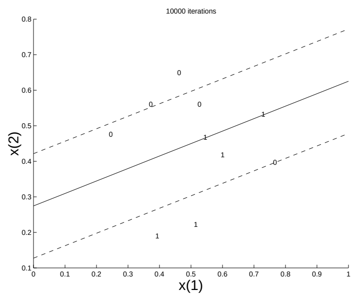

Figure 7: The decision boundary $p(c = 1|\mathbf{x}) = 0.5$ (solid line) and confidence boundaries $p(c = 1|\mathbf{x}) = 0.9$ and $p(c = 1|\mathbf{x}) = 0.1$ for non-linearly separable data. Note how the confidence interval remains broad.

This is negative definite since

$$\sum_{ij} w_i H_{ij} w_j = -\sum_{i,j,\mu} w_i x_i^\mu w_j x_j^\mu \sigma^\mu (1 - \sigma^\mu) = -\left(\sum_i w_i x_i^\mu\right)^2 \sigma^\mu (1 - \sigma^\mu) \tag{1.17}$$

This means that the error surface has a bowl shape, and gradient ascent is guaranteed to find the best solution, provided that the learning rate η is small enough.

Perceptron Convergence Theorem

One can show that, provided that the data is linearly separable, the above procedure used in an online fashion for the perceptron (replacing $\sigma(x)$ with $\theta(x)$) converges in a finite number of steps. The details of this proof are not important for this course, but the interested reader may consult *Neural Networks for Pattern Recognition*, by Chris Bishop. Note that the online version will not converge if the data is not linearly separable. The batch version will converge (provided that the learning rate η is small) since the error surface is bowl shaped.

1.3 Avoiding Overconfident Classification

We saw that in the case that data is linearly separable, the weights will tend to increase indefinitely (unless we use some stopping criterion). One way to avoid this is to penalise weights that get too large. This can be done by adding a penalty term to the objective function $L(\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is a vector of all the parameters, $\boldsymbol{\theta} = (\mathbf{w}, b)$,

$$L'(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) - \alpha \boldsymbol{\theta}^T \boldsymbol{\theta}. \tag{1.18}$$

The scalar constant $\alpha > 0$ encourages smaller values of $\boldsymbol{\theta}$ (remember that we wish to maximise the log likelihood). How do we choose an appropriate value for α? We shall return to this issue in a later chapter on generalisation.

1.4 Logistic Regression and PCA ?

In previous chapters, we have looked at first using PCA to reduce the dimension of the data, so that a high dimensional datapoint $\mathbf{x}$ is represented by a lower dimensional vector $\mathbf{y}$.

If $\mathbf{e}^1, \ldots, \mathbf{e}^m$ are the eigenvectors with largest eigenvalues of the covariance matrix of the high-dimensional data, then the PCA representation is

$$y_i = (\mathbf{e}^i)^T (\mathbf{x} - \mathbf{c}) = (\mathbf{e}^i)^T \mathbf{x} + a_i \tag{1.19}$$

where $\mathbf{c}$ is the mean of the data, and a_i is a constant for each datapoint. Using vector notation, we can write

$$\mathbf{y} = E^T \mathbf{x} + \mathbf{a} \tag{1.20}$$

where E is the matrix who's i^{th} column is the eigenvector $\mathbf{e}^i$. If we were to use logistic regression on the $\mathbf{y}$, the argument of the sigmoid $\sigma(h)$ would be

$$h = \mathbf{w}^T \mathbf{y} + b = \mathbf{w}^T (E^T \mathbf{x} + \mathbf{a}) + b \tag{1.21}$$

$$= (E\mathbf{w})^T \mathbf{x} + b + \mathbf{w}^T \mathbf{a} = \tilde{\mathbf{w}}^T \mathbf{x} + \tilde{b} \tag{1.22}$$

Hence, there is nothing to be gained by first using PCA to reduce the dimension of the data. Mathematically, PCA is a linear projection of the data. The argument of the logistic function is also a linear function of the data, and a linear function combined with another is simply another linear function.

However, there is a subtle point here. If we use PCA first, then use logistic regression afterwards, although overall, this is still representable as a logistic regression problem, the problem is *constrained* since we have forced linear regression to work in the subspace spanned by the PCA vectors. Consider 100 training vectors randomly positioned in a 1000 dimensional space each with a random class 0 or 1. With very high probability, these 100 vectors will be linearly separable. Now project these vectors onto a 10 dimensional space: with very high probability, 100 vectors plotted in a 10 dimensional space will *not* be linearly separable. Hence, arguably, we should not use PCA first since we could potentially transform a linearly separable problem into a non-linearly separable problem.

The XOR problem

Consider the following four training points and class labels
$\{([0,0],0),([0,1],1),([1,0],1),([1,1],0)\}$.

This data represents a basic logic function, the XOR function, and is plotted in fig(8). This function is clearly not representable by a linear decision boundary, an observation much used in the 1960's to discredit work using perceptrons. To overcome this, we clearly need to look at methods with more complex, non-linear decision boundaries – indeed, we encountered a quadratic decision boundary in a previous chapter. Historically, another approach was used to increase the complexity of the decision boundary, and this helped spawn the area of neural networks, to which we will return in a later chapter.

1.5 An Example : Classifying Handwritten Digits

If we apply logistic regression to our often used handwritten digits example, in which there are 300 ones, and 300 sevens in the training data, and the same number in the test data, the training data is found to be linearly separable. This may surprise you, but consider that there are 784 dimensions,

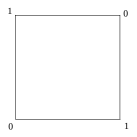

Figure 8: The XOR problem. This is not linearly separable.

and only 600 training points. The stopping criterion used was the same as in the example MATLAB code in this chapter. Using the linear decision boundary, the number of errors made on the 600 test points is 12.

1.6 Support Vector Machines

The realisation that the higher the dimension of the space is, the easier it is to find a hyperplane that linearly separates the data, forms the basis for the Support Vector Machine method. The main idea (contrary to PCA) is to map each vector in a much *higher* dimensional space, where the data can then be linearly separated. Training points which do not affect the decision boundary can then be discarded. We will not go into the details of how to do this in this course, but the interested reader can consult `http://www.support-vector.net`. Related methods currently produce the best performance for classifying handwritten digits – better than average human performance.