Learning from Data
Linear Parameter Models

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

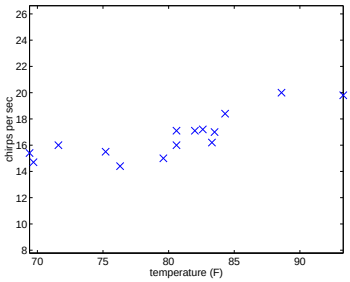

Figure 1: Data from crickets – the number of chirps per second, versus the temperature in Fahrenheit.

1 Introduction

Consider the data in fig(1), in which we plot the number of chirps per second for crickets, versus the temperature in degrees Fahrenheit. A biologist believes that there is a simple relation between the number of chirps and the temperature. Modelling such a relation is a regression problem. The biologist decides to make a straight line fit :

$$c = a + bt \tag{1.1}$$

where she needs to determine the parameters a and b. How can she determine these parameters based on the training data $(c^\mu, t^\mu), \mu = 1, \ldots, 15$? For consistency with our previous notations, let us use y rather than c, and x in place of t, so that our model is $y = a + bx$. The sum squared training error is

$$E(a, b) = \sum_{\mu=1}^{P} (y^\mu - a - bx^\mu)^2 \tag{1.2}$$

Differentiating with respect to a, we find

$$\sum_\mu (y^\mu - a - bx^\mu) = 0 \tag{1.3}$$

Differentiating with respect to b, we find

$$\sum_\mu (y^\mu - a - bx^\mu) x^\mu = 0 \tag{1.4}$$

Dividing by P, we thus have two simultaneous linear equations

$$\langle y \rangle - a - b \langle x \rangle = 0 \tag{1.5}$$

$$\langle yx \rangle - a \langle x \rangle - b \langle x^2 \rangle = 0 \tag{1.6}$$

$$\tag{1.7}$$

where we used the notation $\langle \cdot \rangle$ to denote $\frac{1}{P} \sum_{\mu=1}^{P} \cdot$ .We can easily solve these linear equations to determine a and b. The important thing to note about this regression model is that the *parameters* only appear in a linear fashion. We could also, more conveniently write our model as

$$y = \mathbf{w}^T \boldsymbol{\phi} \tag{1.8}$$

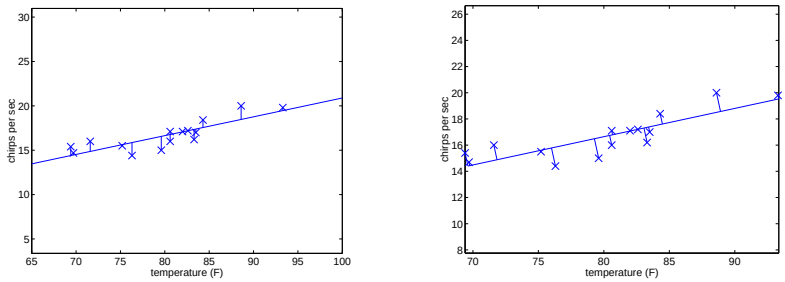

Figure 2: Left: Straight line regression fit to the cricket data. Right: PCA fit to the data. In regression we minimize the residuals – the fit represents the shortest vertical distances. In PCA the fit minimizes the orthogonal projections to the line.

where $\mathbf{w} = (a, b)^T$ and $\boldsymbol{\phi} = (1, x)^T$. The training error then is

$$E(\mathbf{w}) = \sum_{\mu=1}^{P} (y^\mu - \mathbf{w}^T \boldsymbol{\phi}^\mu)^2 \tag{1.9}$$

where $\boldsymbol{\phi}^\mu = (1, x^\mu)^T$. We now wish to determine the parameter vector $\mathbf{w}$. Writing out the error in terms of the components,

$$E(\mathbf{w}) = \sum_{\mu=1}^{P} (y^\mu - \sum_i w_i \phi_i^\mu)(y^\mu - \sum_j w_j \phi_j^\mu) \tag{1.10}$$

Differentiating with respect to w_k, this gives

$$\sum_\mu y^\mu \phi_k^\mu = \sum_i w_i \sum_\mu \phi_i^\mu \phi_k^\mu \tag{1.11}$$

or, in matrix notation,

$$\sum_\mu y^\mu \boldsymbol{\phi}^\mu = \sum_\mu \boldsymbol{\phi}^\mu (\boldsymbol{\phi}^\mu)^T \mathbf{w} \tag{1.12}$$

Hence, the solution is

$$\mathbf{w} = \left(\sum_\mu \boldsymbol{\phi}^\mu (\boldsymbol{\phi}^\mu)^T \right)^{-1} \sum_\mu y^\mu \boldsymbol{\phi}^\mu \tag{1.13}$$

Putting in the actual data, we get $a = -0.3091$, $b = 0.2119$. The fit is plotted in fig(2). Although the solution is written in terms of the inverse matrix, we never actually compute the inverse numerically; we use instead Gaussian elimination – see the MATLAB code.

1.1 Regression and PCA

In an earlier chapter, we discussed using PCA to reduce the dimensionality of data, based on the idea that data may lie close to a low dimensional hyperplane. Since a line is a low dimensional hyperplane, one may wonder

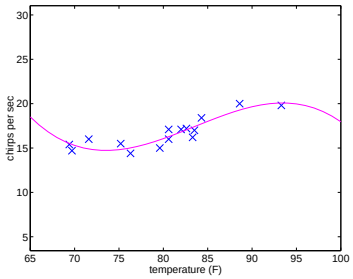

Figure 3: Cubic polynomial fit to the cricket data.

what the difference is between using PCA to fit a line and the above regression approach. The answer is that the objective functions are different. Regression finds a line that minimizes the vertical distance between a datapoint and the line; PCA finds a line that minimizes the distance between a datapoint and the line – see fig(2).

2 Linear Parameter Models (Generalised Linear Models)

A linear parameter model is defined as

$$y(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) \tag{2.1}$$

As we saw above, straight line regression fits to data are examples of this. If we choose the coefficients of the vector $\boldsymbol{\phi}$ to be non-linear functions of $\mathbf{x}$, then the mapping $\mathbf{x} \rightarrow y$ will be non-linear. The phrase "linear" model here refers to the fact that the model depends on its *parameters* in linear way. This is an extremely important point. Unfortunately, the terminology is a little confused in places. These models are often referred to as "generalised linear models". However, sometimes people use this same phrase to refer to something completely different – beware!

2.1 Training LPMs

In the derivation above, there was nothing specific about the form of $\boldsymbol{\phi}$. Hence, the solution in equation (2.5) holds in general. That is, you simply put in a different $\boldsymbol{\phi}$ vector if you wish to find a new solution. For example, consider the case of fitting a cubic function $y = w_1 + w_2 x + w_3 x^2 + w_4 x^3$ to the data. In this case, we would choose

$$\boldsymbol{\phi} = \left(1, x, x^2, x^3\right)^T \tag{2.2}$$

The solution has the same form, except $\mathbf{w}$ is now a 4 dimensional vector

```
% Linear Parameter Regression : example using Cubic Polynomial
data=[
   20.0 16.0 19.8 18.4 17.1 15.5 14.7 17.1 15.4 16.2 15.0 17.2 16.0 17.0 14.4 ;
   88.6 71.6 93.3 84.3 80.6 75.2 69.7 82.0 69.4 83.3 79.6 82.6 80.6 83.5 76.3];

x = data(2,:); y = data(1,:); plot(x,y,'x')

phi = phi_fn(x); n = size(phi,1);
w = (phi*phi' + 10^(-7)*eye(n))\sum(repmat(y,size(phi,1),1).*phi,2); % Better than inv

xp = 65:100; yp = w'*phi_fn(xp); hold on; plot(xp,yp)

function phi = phi_fn(x)
  phi(1,:) = ones(1,length(x)); % cubic polynomial :
  phi(2,:) = x;
  phi(3,:) = x.^2;
  phi(4,:) = x.^3;
```

The above MATLAB code implements LPM in general. All that needs to be changed in the above code for a different model is the function `phi_fn`. Note that, rather than using the `inv` function in MATLAB to solve the linear equations, it is much better to use the slash function \ – this implements Gaussian elimination to solve linear systems. This is both much faster and more accurate. As a rule we never invert matrices unless you need to – and you never need to if you only want to solve the linear system.

Choosing between Different Models

How would we decide if a straight line fit is preferable to a cubic polynomial fit? A general way to address this problem is to use some validation data to test how accurate each model predicts the validation data. The more accurate model on the validation data would then be preferred.

2.2 Regularisation and numerical stability

It should be fairly clear from the above that all polynomial regression fits are simply special cases of LPMs. Also, the more terms there are in polynomial, the more curved can be the fit to the data. One way to penalise too complex models is to use a penalty term

$$E_{reg}(\mathbf{w}) = \mathbf{w}^T\mathbf{w} \tag{2.3}$$

The regularised training error is then

$$E_{regtrain}(\mathbf{w}, \lambda) = \sum_{\mu=1}^{P}(y^\mu - \mathbf{w}^T\boldsymbol{\phi}^\mu)^2 + \lambda\mathbf{w}^T\mathbf{w} \tag{2.4}$$

If we differentiate the regularised training error to find the optimal $\mathbf{w}$ for a given λ, we find: Hence, the solution is

$$\mathbf{w} = \left(\sum_\mu \boldsymbol{\phi}^\mu(\boldsymbol{\phi}^\mu)^T + \lambda\mathrm{I}\right)^{-1}\sum_\mu y^\mu\boldsymbol{\phi}^\mu \tag{2.5}$$

where I is the $n \times n$ identity matrix and $n = dim(\mathbf{w})$. Another beneficial aspect of using a quadratic penalty term is that the solution is more numerically stable – this can be a problem in cases where there is limited training data. We can determine λ by using a validation set.

2.3 Higher Dimensional Outputs

It is straightforward to generalise the above framework to cases where there is more than one output variable – rather there is an output vector $\mathbf{y}$:

$$y_i(\mathbf{x}) = \mathbf{w}_i^T \boldsymbol{\phi}(\mathbf{x}) \tag{2.6}$$

The mathematics follows similarly to before, and this is left as an exercise for the interested reader.

2.4 Classification

One way to adapt the LPM model to classification is to use $p(c = 1|\mathbf{x}) = \sigma(\mathbf{w}^t \boldsymbol{\phi}(\mathbf{x}))$. The logistic regression model simply used a special case in which the vector $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{x}$. However, there is nothing to stop us using this more general method. The nice thing is that the decision boundary is then a non-linear function of $\mathbf{x}$. Clearly, instead of using the euclidean square distance as the error measure, we now use the log-likelihood, exactly as in the chapter on logistic regression. Again, however, the training to find $\mathbf{w}$ will not be so straightforward, since the obejective function is not quadratic. However, the surface remains well behaved so that finding a solution is not numerically difficult. We leave it as an exercise for the reader to work out the details.

3 Radial Basis Functions

A popular choice for the vector $\boldsymbol{\phi}(\mathbf{x})$ is the radial basis function :

$$\phi_i(\mathbf{x}) = e^{-\frac{1}{2\alpha^2}(\mathbf{x}-\mathbf{m}^i)^2} \tag{3.1}$$

where the vectors $\mathbf{m}^i, i = 1, \ldots, M$ define M centres in the input space. The parameter α determines the width of each basis function. These basis functions are bump shaped, with the position of the bump being given by m and the width by α. An example is given in fig(4)(Left) in which several RBFs are plotted. In regression, we can then use a linear combination of these "bumps" to fit the data. For example consider fitting the data in fig(4)(Right).

Setting α We use the validation data to set α. Throughout these experiments, I set the regularisation parameter $\lambda = 0.0001$. In principle one could use the validation set to optimise over both α and λ. In fig(5) we plot the validation error as a function of α. Based on this graph, we can find the best value of α; that which minimises the validation error. The predictions are also given in fig(5).

4 The curse of Dimensionality

We saw that using radial basis functions we can get good predictions, provided that we choose appropriate basis functions (set the widths correctly).

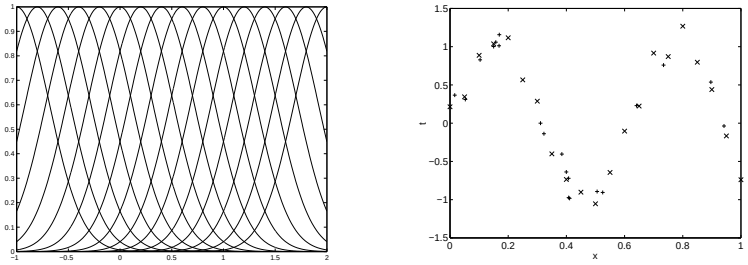

Figure 4: Left: A set of radial basis functions, $\alpha = 5$, with $m = -1, -0.8, -0.6, \ldots, 2$. Right: Data to be fitted. The $\times$ are the training points, and the $+$ are the validation points.

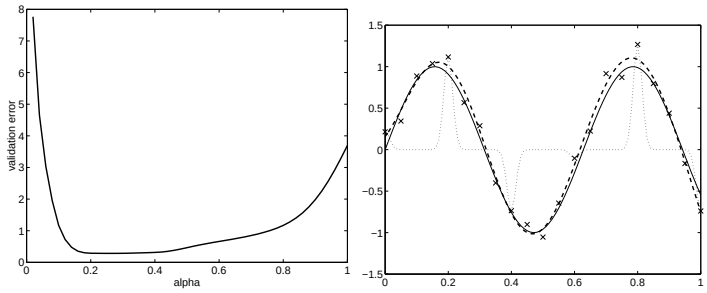

Figure 5: Left: The validation error as a function of the basis function width. Right: The predictions. The solid line is the correct underlying function $\sin(10x)$; the dashed line is the best predictor based on the validation set. The dotted line is the worst predictor based on the validation set.

It seems intuitively clear that if the data has non-trivial behaviour over some region in x, then we need to cover the region of x space fairly densely with "bump" type functions. In the above case, we used 16 basis functions for this one dimensional space. In 2 dimensions, we can also use bump type functions. However, we now need to cover a 2 dimensional space. If we wish to cover it to the same discretisation level, we would need $16^2 = 256$ basis functions. In an n dimensional input space, we would need 16^n functions. This is an extremely rapidly growing function of n so that in 10 dimensions, we would need $16^{10} = 10^{12}$ basis functions. This means we would have to solve linear systems in 10^{12} variables! This cannot be easily done. This explosion in the apparent number of basis functions required is the famous "curse of dimensionality".

A possible solution is to make the basis functions very broad to cover more of the high dimensional space. However, this will mean a lack of flexibility of the fitted function. Another approach is to place basis functions centred on the training input points that we have, and add some more basis functions randomly placed close to the training inputs. The rational behind this is that when we come to do prediction, we will most likely see novel $\mathbf{x}$ that are close to the training points – we do not need to make "accurate" predictions over all the space.

A further approach is to make the positions of the basis functions adaptive,

allowing them to be moved around in the space to minimise the error. This approach motivates the neural network models.

The criticism of the curse of dimensionality is, in my humble opinion, rather weak, and good results are often obtained by using basis functions which are dense around the training inputs.

5 Summary

- Linear Parameter models are regression models that are linear in the parameters.
- They are very easy to train (no local minima).
- Criticised in high dimensional input spaces due to the curse of dimensionality.
- Judicious placement of the basis functions on close to the training inputs is a workaround for the curse of dimensionality. Otherwise we need to optimise the placement of the basis functions – that is, use neural networks.
- Easily adapted to classification (though the training is now more difficult and needs to be solved using optimisation).