Chapter 1

Linear Modelling: A Least Squares Approach

An important and general problem in Machine Learning, which has wide application, is *learning* or *inferring* a functional relationship between a set of *attribute* variables and associated *response* or *target* variables so that we can predict the response for any set of attributes. For example, we may wish to build a model that can perform disease diagnosis. To do this we would use a dataset comprised of measurements (attributes; e.g. blood pressure, heart rate, weight etc) taken from patients with known disease state (responses; healthy or diseased). In a completely different example, we may wish to make recommendations to customers. In this case, we could build a **model** from descriptors of items a particular customer had previously bought (attributes) and whether or not the customer ultimately liked the product (response). This would enable us to predict which objects a customer would like and hence make recommendations. There are many more important application areas that we will come across throughout this text.

1.1 Linear modelling

To begin with we will consider, using a practical example, the most straightforward of *learning* problems, linear modelling [1] – learning a **linear** relationship between attributes and responses. Figure 1.1 shows the gold medal winning time for the men's 100 m at each of the Olympic Games held since 1896. Our aim is to use this data to *learn* a model of the functional dependence (if one exists) between Olympic year and 100 m winning time and use this model to make predictions about the winning times in future games. Clearly the year is not the only factor that affects the winning time and if we were interested in using our predictions seriously we may want to take other things into account (the recent form of the main competitors is an obvious example). However, examining Figure 1.1 we can see that there is at least a statistical dependence between year and winning time (it may not be a *causal* dependence – elapsing years are not directly *causing* the drop in winning times) and this is enough to help us introduce and develop the main ideas of linear modelling.

1.1.1 Defining the model

We can begin by defining our model as a function which maps our *input* attributes, in this case Olympic year, to our *output* or target values – winning time. For our attributes, we will use the numerical value of the year – e.g.

[1] The type of modelling we will consider here is often known as *regression* and was originally used in the context of genetics by Francis Galton (1877) when studying how intelligence is passed on (or not as the case may be) from generation to generation. The term was then adopted by statisticians who developed Galton's work within a statistical context.

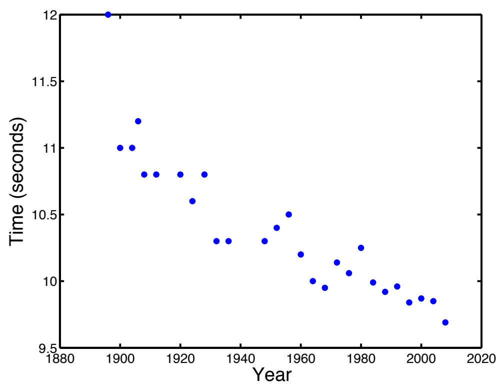


FIGURE 1.1: Winning men's 100 m times at the Summer Olympics since 1896. Note that the two world wars interrupt the games in 1914, 1940 & 1944.

1980 – although there are alternative formulations (e.g. years since the first modern games, $1980 - 1896 = 84$) that would make no real difference to the underlying assumptions.

There are many **functions** that could be used to define this mapping. In general, this function will take as an input x (the Olympic year) and will return t (the winning time in seconds). In other words, t is a function of x. Mathematically, we will write this as $t = f(x)$. In some cases, all we will need to know to evaluate our function is x. For example, if $f(x) = \sin(x)$ or, say $f(x) = x$ we can compute t for any x. In general, we will need to be more flexible and it is likely that our model will have a set of associated **parameters**. For example, $t = ax$ has a parameter called a that needs to be defined somehow. Learning model parameters from a suitable dataset is a common theme in Machine Learning. We will use $t = f(x; a)$ to denote a function $f(\cdot)$ that acts on x and has a parameter a.

Comment 1.1 – Linear relationships: The equation:

$$y = mx + c,$$

where m and c are constant, defines a linear relationship between x and y. It is called linear because the relationship between x and y could be visualised as a straight line. The following equations are nonlinear due to the more complex forms in which we find the variables x and y:

$$y = mx^2 + c, \quad y = \sin(x), \quad \sqrt{y} = mx + c.$$

The values of m and c to not effect the linearity of the relationship. For example, the following still represent linear relationships between x and y:

$$y = mx + c^2, \quad y = x\sin(m) + c.$$

1.1.2 Modeling assumptions

To help us choose which particular model to use, we need to make some assumptions. Our principal assumption at this stage is the following:

The relationship between x and t is linear (see Comment 1.1).

This could be stated alternatively as:

The data in Figure 1.1 could be adequately modelled with a straight line.

Or:

The winning time drops by the same amount every M years.

Examining Figure 1.1 we can see that this assumption is not perfectly satisfied. However, it is our hope that it is adequate and will produce a model that is useful in the sense that it can make predictions regarding winning times in the future.

The simplest model that satisfies our assumptions is

$$t = f(x) = x,$$

the winning time is equal to the Olympic year. The fact that x takes values greater than or equal to 1880 and t values less than or equal to 12, and that the winning time is decreasing as the year increases tells us that this model is inadequate. Adding a single parameter results in:

$$t = f(x; w) = wx,$$

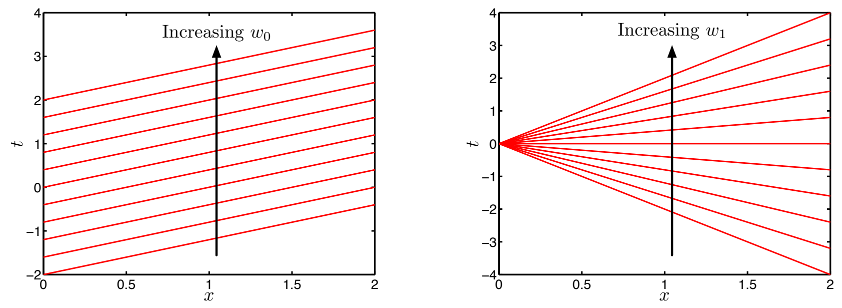
where w can be either positive or negative. This enhanced model lets us produce a straight line with any gradient through the choice of w. This is an increase in flexibility but it is still limited by the fact that at year 0, the model predicts a winning time of $w \times 0 = 0$. Looking at the data we can see that this is not realistic – following the general trend of the data, the winning time at year 0 is actually going to be quite a large number. Adding one more parameter to the model overcomes this limitation:

$$t = f(x; w_0, w_1) = w_0 + w_1 x, \tag{1.1}$$

This is the standard equation for a straight line that many readers will have encountered before. The *learning* task now involves using the data in Figure 1.1 to choose suitable values for the two parameters w_0 and w_1. These two parameters are often known as the intercept (w_0; where the line intercepts the t-axis) and the gradient (w_1; the gradient of the line) and the effect of varying them can be seen in Figure 1.2 (Matlab script: `plotlinear.m`) (see Exercise EX 1.1).

1.1.3 Defining what a *good* model is

In order to choose values of w_0 and w_1 that are somehow *best*, we need to define what *best* means. Common sense would suggest that the *best* solution consists of the values of w_0 and w_1 that produce a line that passes as close as possible to *all* of the data points. A common way of measuring how close a particular model gets to one



(a) Increasing w_0 changes the point at which the line crosses the t axis.

(b) Increasing w_1 changes the gradient of the line.

FIGURE 1.2: Effect of varying w_0 and w_1 in the linear model defined by Equation 1.1.

of the data points is the squared difference between the true winning time and the winning time predicted by the model. Using x_n, t_n to denote the nth Olympic year and winning time respectively, the squared difference is defined as:

$$(t_n - f(x_n; w_0, w_1))^2.$$

The smaller this number is, the closer the model, at x_n, is to t_n. Squaring the difference is important. Without it, we could indefinitely reduce this quantity by continually increasing $f(x_n; w_0, w_1)$.

This expression is known as the *squared loss function* as it describes how much accuracy we are losing through the use of $f(x_n; w_0, w_1)$ to model t_n. Throughout this text, we will use $\mathcal{L}_n()$ to denote loss functions. In this case,

$$\mathcal{L}_n(t_n, f(x_n; w_0, w_1)) = (t_n - f(x_n; w_0, w_1))^2 \tag{1.2}$$

is the loss for year n. Loss is always positive and the lower the *loss*, the better our function describes the data. As we want a low *loss* for all of the N years, we consider the *average loss* across the whole dataset, given as

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_n(t_n, f(x_n; w_0, w_1)). \tag{1.3}$$

This is the average of the loss values at each of the N years. The lower it is, the better. We will therefore tune w_0 and w_1 to produce the model that results in the lowest value of the average loss, $\mathcal{L}$. Finding these *best* values for w_0 and w_1 can be expressed mathematically as

$$\underset{w_0, w_1}{\operatorname{argmin}} \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_n(t_n, f(x_n; w_0, w_1)).$$

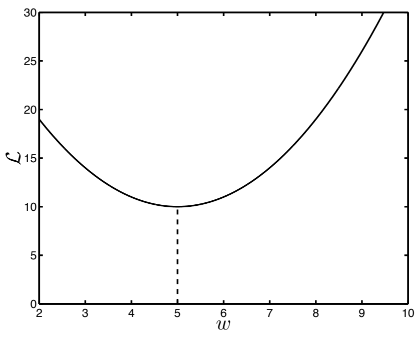


FIGURE 1.3: Example loss function of one parameter (w). The dashed line shows the value of w that minimised the loss ($w = 5$).

The term argmin is the mathematical shorthand for 'find the argument that minimises...'. In this instance, the argument(s) are the values of w_0 and w_1 and the expression to be minimised is the average loss. Figure 1.3 shows a hypothetical loss that is a function of a single parameter, w. The value of w that minimises $\mathcal{L}$ is $w = 5$. Historically, minimisation of the squared loss is the basis of the *Least-Squares* errors method of function approximation which dates back to methods developed by Gauss and Legendre (1809) when predicting planetary motion.

Other loss functions exist that are suitable for regression. For example, a common alternative is the absolute loss:

$$\mathcal{L}_n = |t_n - f(x_n; w_0, w_1)|.$$

The squared loss is a very common choice, in part due to the fact that it makes finding the best values of w_0 and w_1 relatively straightforward – we can derive an **analytical** solution. However, modern computational power has reduced the importance of mathematical convenience – there is no longer any excuse for choosing a convenient loss function over one more suited to the data. This notwithstanding, our aim is to introduce general modelling concepts for which the squared loss will be adequate. It is worth bearing in mind that others are available and, in many cases, will be more appropriate.

1.1.4 The least squares solution – a worked example

To recap, our dataset consists of $n = 1, \ldots, N$ observations, each of which consists of a year x_n and a time in seconds t_n.

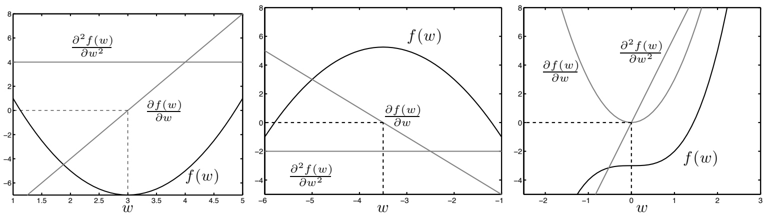
We are going to attempt to find a functional relationship using a linear model defined as

$$f(x; w_0, w_1) = w_0 + w_1 x \tag{1.4}$$

and we have decided that we will use the least squares loss function to choose suitable values of w_0 and w_1. Substituting the linear model into the expression for average loss and multiplying out the brackets results in

$$\begin{aligned}
\mathcal{L} &= \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}_n(t_n, f(x_n; w_0, w_1)) \\
&= \frac{1}{N}\sum_{n=1}^{N}(t_n - f(x_n; w_0, w_1))^2 \\
&= \frac{1}{N}\sum_{n=1}^{N}(t_n - (w_0 + w_1 x_n))^2 \\
&= \frac{1}{N}\sum_{n=1}^{N}(w_1^2 x_n^2 + 2w_1 x_n w_0 - 2w_1 x_n t_n + w_0^2 - 2w_0 t_n + t_n^2) \\
&= \frac{1}{N}\sum_{n=1}^{N}(w_1^2 x_n^2 + 2w_1 x_n (w_0 - t_n) + w_0^2 - 2w_0 t_n + t_n^2) \qquad (1.5)
\end{aligned}$$

Comment 1.2 – Turning points: We can find turning points (that might correspond to minima) of a function, $f(w)$ by searching for points where the gradient of the function, $\frac{\delta f(w)}{\delta w}$ is zero. To determine whether or not a turning point corresponds to a maximum, minimum or saddle point, we can examine the second derivative – $\frac{\delta^2 f(w)}{\delta w^2}$. If, at a turning point $\hat{w}$, the second derivative is positive, we know that this turning point is a minimum. The following 3 plots show three example functions along with their first and second derivatives:



In general, a function may have several turning points. An interesting special case is functions whose second derivative is a positive constant – these correspond to functions that have only one minimum.

Differentiating the loss function: At a minimum of $\mathcal{L}$, the **partial derivatives** with respect to w_1 and w_0 must be zero (see Comment 1.2). Therefore, computing the partial derivatives, equating them to zero and solving for w_0 and w_1 will give us a potential minimum. Starting with w_1, we know that terms in Equation 1.5 that do not include w_1 can be ignored (as their partial derivative with respect to w_1 will be zero). Removing these terms leaves

$$\frac{1}{N}\sum_{n=1}^{N}[w_1^2 x_n^2 + 2w_1 x_n w_0 - 2w_1 x_n t_n].$$

Before we take the partial derivatives, we will re-arrange the expression to make it a little simpler. In particular, taking terms that are not indexed by n outside of the sum and re-arranging results in

$$w_1^2 \frac{1}{N}\left(\sum_{n=1}^N x_n^2\right) + 2w_1 \frac{1}{N}\left(\sum_{n=1}^N x_n(w_0 - t_n)\right).$$

Taking the partial derivative with respect to w_1 gives us the following expression:

$$\frac{\partial \mathcal{L}}{\partial w_1} = 2w_1 \frac{1}{N}\left(\sum_{n=1}^N x_n^2\right) + \frac{2}{N}\left(\sum_{n=1}^N x_n(w_0 - t_n)\right). \quad (1.6)$$

Now we do the same for w_0. Removing non w_0 terms leaves:

$$\frac{1}{N}\sum_{n=1}^N [w_0^2 + 2w_1 x_n w_0 - 2w_0 t_n].$$

Again, we will re-arrange it a bit before we differentiate. Moving terms not indexed by n outside of the summation (noting that $\sum_{n=1}^N w_0^2 = N w_0^2$) results in

$$w_0^2 + 2w_0 w_1 \frac{1}{N}\left(\sum_{n=1}^N x_n\right) - 2w_0 \frac{1}{N}\left(\sum_{n=1}^N t_n\right).$$

Taking the partial derivative with respect to w_0 results in:

$$\frac{\partial \mathcal{L}}{\partial w_0} = 2w_0 + 2w_1 \frac{1}{N}\left(\sum_{n=1}^N x_n\right) - \frac{2}{N}\left(\sum_{n=1}^N t_n\right). \quad (1.7)$$

Equating the derivatives to zero: We now have expressions for the partial derivatives of the loss with respect to both w_0 and w_1. To find the values of w_0 and w_1 that correspond to a turning point (hopefully a minimum), we must set these expressions to zero and solve for w_0 and w_1. It's easiest to start with the expression for w_0. Setting Equation 1.7 to zero and solving for w_0:

$$\begin{aligned} 2w_0 &+ 2w_1 \frac{1}{N}\left(\sum_{n=1}^N x_n\right) - \frac{2}{N}\left(\sum_{n=1}^N t_n\right) = 0 \\ 2w_0 &= \frac{2}{N}\left(\sum_{n=1}^N t_n\right) - w_1 \frac{2}{N}\left(\sum_{n=1}^N x_n\right) \\ w_0 &= \frac{1}{N}\left(\sum_{n=1}^N t_n\right) - w_1 \frac{1}{N}\left(\sum_{n=1}^N x_n\right). \end{aligned}$$

Denoting the average winning time as $\bar{t} = \frac{1}{N}\sum_{n=1}^N t_n$ and the average Olympic year as $\bar{x} = \frac{1}{N}\sum_{n=1}^N x_n$ we can rewrite our expression for the value of w_0 at the turning point ($\widehat{w_0}$) as

$$\widehat{w_0} = \bar{t} - w_1 \bar{x}. \quad (1.8)$$

What insight can we gain from this expression? This new expression is a rearrangement of our original model ($t_n = w_0 + w_1 x_n$) where t_n and x_n have been replaced by their average values $\bar{t}$ and $\bar{x}$. Consider the value of our function averaged over the N data points. This is given by:

$$\frac{1}{N}\sum_{n=1}^{N} f(x_n; w_0, w_1) = \frac{1}{N}\sum_{n=1}^{N}(w_0 + w_1 x_n) = w_0 + w_1\bar{x}.$$

The average winning time is given by $\bar{t}$ so, in using Equation 1.8, we are choosing $\widehat{w_0}$ to ensure that the average value of the function is equal to the average winning time. Intuitively, matching the averages in this way seems very sensible.

Before we use Equation 1.6 to get an expression for $\widehat{w_1}$ (the value of w_1 at the turning point – see Comment 1.2), it is worth briefly examining the second derivatives to ensure that this is a minimum. Differentiating Equation 1.6 again with respect to w_1 and Equation 1.7 again with respect to w_0 results in:

$$\begin{aligned}\frac{\delta^2\mathcal{L}}{\delta w_1^2} &= \frac{2}{N}\sum_{n=1}^{N} x_n^2 \\ \frac{\delta^2\mathcal{L}}{\delta w_0^2} &= 2. \end{aligned} \tag{1.9}$$

Both of these quantities must be positive. This tells us that there will be only one turning point and it will correspond to a minimum of the loss.

This process has supplied us with an expression for the value of $\widehat{w_0}$ – the value of w_0 that minimises the loss. This expression depends on w_1 implying that for any particular w_1, we know the best w_0. Substituting our expression for the best w_0 value (Equation 1.8) into Equation 1.6 and rearranging, we obtain an expression that only includes w_1 terms:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial w_1} &= w_1\frac{2}{N}\left(\sum_{n=1}^{N} x_n^2\right) + \frac{2}{N}\left(\sum_{n=1}^{N} x_n(\widehat{w_0} - t_n)\right) \\ &= w_1\frac{2}{N}\left(\sum_{n=1}^{N} x_n^2\right) + \frac{2}{N}\left(\sum_{n=1}^{N} x_n(\bar{t} - w_1\bar{x} - t_n)\right) \\ &= w_1\frac{2}{N}\left(\sum_{n=1}^{N} x_n^2\right) + \bar{t}\frac{2}{N}\left(\sum_{n=1}^{N} x_n\right) - w_1\bar{x}\frac{2}{N}\left(\sum_{n=1}^{N} x_n\right) - \frac{2}{N}\left(\sum_{n=1}^{N} x_n t_n\right).\end{aligned}$$

We can simplify this expression by using $\bar{x} = (1/N)\sum_{n=1}^{N} x_n$ as before and gathering together w_1 terms:

$$\frac{\partial\mathcal{L}}{\partial w_1} = 2w_1\left[\left(\frac{1}{N}\sum_{n=1}^{N} x_n^2\right) - \bar{x}\bar{x}\right] + 2\bar{t}\bar{x} - 2\frac{1}{N}\left(\sum_{n=1}^{N} x_n t_n\right).$$

Finally, we can get an expression for $\widehat{x_1}$ by setting this partial derivative to zero and solving for w_1:

$$\begin{aligned}
2w_1\left[\left(\frac{1}{N}\sum_{n=1}^{N}x_n^2\right)-\bar{x}\bar{x}\right] &+ 2\bar{t}\bar{x}-2\frac{1}{N}\left(\sum_{n=1}^{N}x_nt_n\right)=0\\
2w_1\left[\left(\frac{1}{N}\sum_{n=1}^{N}x_n^2\right)-\bar{x}\bar{x}\right] &= 2\frac{1}{N}\left(\sum_{n=1}^{N}x_nt_n\right)-2\bar{t}\bar{x}\\
\widehat{w_1} &= \frac{\frac{1}{N}\left(\sum_{n=1}^{N}x_nt_n\right)-\bar{t}\bar{x}}{\left(\frac{1}{N}\sum_{n=1}^{N}x_n^2\right)-\bar{x}\bar{x}}.
\end{aligned}$$

It is helpful to now define some new average quantities. The first, $(1/N)\sum_{n=1}^{N}x_n^2$ is the average squared value of the data and we will denote this $\overline{x^2}$. Note that this quantity is not the same as $(\overline{x})^2$. The second is $(1/N)\sum_{n=1}^{N}x_nt_n$ (which, similarly is not the same as $\bar{x}\bar{t}$). We will denote this as $\overline{xt}$. Substituting these into our expression for w_1 gives:

$$\widehat{w_1}=\frac{\overline{xt}-\bar{x}\bar{t}}{\overline{x^2}-(\bar{x})^2} \qquad (1.10)$$

Equations 1.10 and 1.8 provide everything required to compute the best parameter values. Firstly $\widehat{w_1}$ from Equation 1.10 which is then substituted into Equation 1.8 to calculate $\widehat{w_0}$ (Matlab script: `fitlinear.m`).

1.1.5 Worked example

Before we fit the linear model to the Olympic data, it is useful to provide a worked example on a smaller dataset. Assume we observe $N = 3$ data-points, provided in Table 1.1. The final row also gives the various averages required to compute $\widehat{w_0}$ and $\widehat{w_1}$: $\bar{x}$, $\bar{t}$, $\overline{xt}$ and $\overline{x^2}$. The three data points are plotted in Figure 1.4.

n	x_n	t_n	x_nt_n	x_n^2
1	1	4.8	4.8	1
2	3	11.3	33.9	9
3	5	17.2	86	25
$(1/N)\sum_{n=1}^{N}$	3	11.1	41.57	11.67

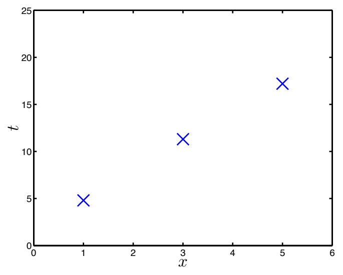
TABLE 1.1: Synthetic dataset for linear regression example.

Substituting these values into Equation 1.10 gives:

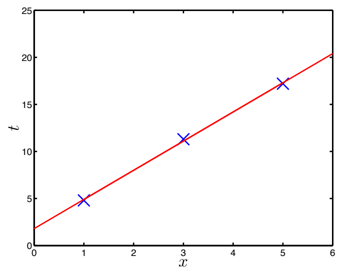
$$\begin{aligned}
w_1 &= \frac{41.57-3\times 11.1}{11.67-3\times 3}\\
&= \frac{8.27}{2.67}\\
&= 3.1
\end{aligned}$$

and:

$$w_0 = 11.1 - 3.1\times 3 = 1.8.$$



(a) The three synthetic data points described in Table 1.1



(b) The least squares fit defined by $f(x; w_0, w_1) = 1.8 + 3.1x$

FIGURE 1.4: Data and function for the worked example of Section 1.1.5.

Our best linear function is therefore:

$$f(x; w_0, w_1) = 1.8 + 3.1x,$$

and it is shown in Figure 1.4(b).

1.1.6 Least squares fit to the Olympic data

The data for the Olympic 100 m dataset (shown in Figure 1.1) is summarised in Table 1.2. Applying exactly the same methodology to this data, we obtain the following values for w_1 and w_0 (note that our final values were worked out in Matlab – if you work through, you might get slightly different results due to rounding errors):

$$\begin{aligned} w_1 &= \frac{20268.1 - 1952.37 \times 10.39}{3.8130 \times 10^6 - 1952.37 \times 1952.37} \\ &= \frac{-16.3}{1225.5} \\ &= -0.0133 \\ w_0 &= 10.39 - (-0.0133) \times 1952.37 \\ &= 36.416. \end{aligned}$$

Therefore, our best linear function is:

$$f(x; w_0, w_1) = 36.416 - 0.013x. \tag{1.11}$$

The function is plotted in Figure 1.5 (see Exercise EX 1.2). Do these values agree with the approximations you made in Exercise EX 1.1? (Matlab script: `fitolympic.m`)

1.1.7 Summary

It is worth recapping the topics covered so far. We have introduced the idea of creating a model (in particular a linear one) that encapsulates the relationship

n	x_n	t_n	x_nt_n	x_n^2
1	1896	12.00	22752.0	3.5948×10^6
2	1900	11.00	20900.0	3.6100×10^6
3	1904	11.00	20944.0	3.6252×10^6
4	1906	11.20	21347.2	3.6328×10^6
5	1908	10.80	20606.4	3.6405×10^6
6	1912	10.80	20649.6	3.6557×10^6
7	1920	10.80	20736.0	3.6864×10^6
8	1924	10.60	20394.4	3.7018×10^6
9	1928	10.80	20822.4	3.7172×10^6
10	1932	10.30	19899.6	3.7326×10^6
11	1936	10.30	19940.8	3.7481×10^6
12	1948	10.30	20064.4	3.7947×10^6
13	1952	10.40	20300.8	3.8103×10^6
14	1956	10.50	20538.0	3.8259×10^6
15	1960	10.20	19992.0	3.8416×10^6
16	1964	10.00	19640.0	3.8573×10^6
17	1968	9.95	19581.6	3.8730×10^6
18	1972	10.14	19996.1	3.8888×10^6
19	1976	10.06	19878.6	3.9046×10^6
20	1980	10.25	20295.0	3.9204×10^6
21	1984	9.99	19820.2	3.9363×10^6
22	1988	9.92	19721.0	3.9521×10^6
23	1992	9.96	19840.3	3.9681×10^6
24	1996	9.84	19640.6	3.9840×10^6
25	2000	9.87	19740.0	4.0000×10^6
26	2004	9.85	19739.4	4.0160×10^6
27	2008	9.69	19457.5	4.0321×10^6
$(1/N)\sum_{n=1}^{N}$	1952.37	10.39	20268.1	3.8130×10^6

TABLE 1.2: Olympic men's 100 m data.

between a set of attributes and a set of responses. To enable us to fit (or learn) this model from data, we defined a *loss* function as a way of objectively identifying how good a particular model was. Using the squared loss, we derived exact expressions for the values of the model parameters that minimised the loss and therefore corresponded to the best function. Finally, we applied this technique to two different data sets. We shall now see how we can use the model to make predictions.

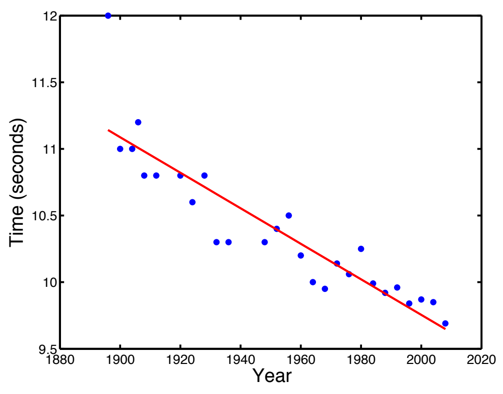


FIGURE 1.5: The least squares fit ($f(x; w_0, w_1) = 36.416 - 0.013x$) to the men's 100 m Olympic dataset.

1.2 Making predictions

Now that we have a model relating the Olympic year to the winning 100 m sprint time, we can use it to predict the winning time for a year that we have not yet observed. For example, to predict the winning times at the 2012 and 2016 Olympics, t^{2012} and t^{2016}, we plug $x = 2012$ and $x = 2016$ into our formula.

$$\begin{aligned} f(x; w_0 = 36.416, w_1 = -0.0133) &= 36.416 - 0.0133x \\ t^{2012} = f(2012; w_0, w_1) &= 36.416 - 0.0133 \times 2012 = 9.595\ s \\ t^{2016} = f(2016; w_0, w_1) &= 36.416 - 0.0133 \times 2016 = 9.541\ s \end{aligned}$$

These predictions can be seen in Figure 1.6 (Matlab script: `olymppred.m`). They tell us that based on our linear regression model we might expect a winning time of 9.595 s in London in 2012. This value is very precise. It seems unlikely that *any* model would be able to predict the outcome of such a complex event to such a high degree of accuracy, least of all one based on nothing more than a straight line. Our model is not even able to predict data that it has seen very precisely, as can be seen by the distance of some points to the line in Figure 1.5. Assuming that it will become more precise into the future seems particularly foolish.

Precise predictions are only of limited use in situations where our model is not perfect (almost all situations). In general, it is more useful to be able to express a range of values rather than any particular one. We shall see how to do this in Chapter 2 and beyond.

1.2.1 A second Olympic dataset

A second dataset, related to the first, is shown in Table 1.3 and is plotted, along with the linear model that minimises the squared loss in Figure 1.7 (see Exercise EX

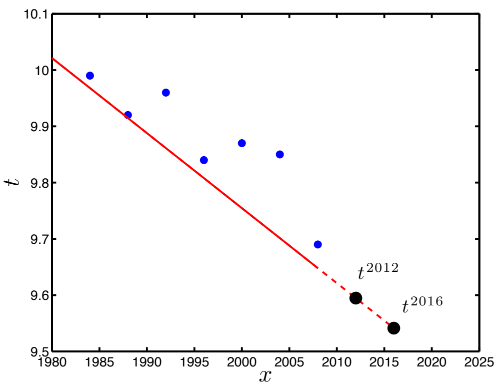


FIGURE 1.6: Zoomed in plot of the winning time in the Olympic men's 100 m sprint from 1980 showing predictions for both the 2012 and 2016 Olympics.

1.6 and Exercise EX 1.7). The model for the Women's data is (remember that you might find slight differences to these values due to rounding errors):

$$f(x; w_0, w_1) = 40.92 - 0.015x.$$

It is interesting to compare this with the model obtained for the men's data:

$$f(x; w_0, w_1) = 36.416 - 0.013x.$$

The women's model has a higher intercept (w_0) and a steeper negative gradient (w_1). If we plot the two models together, as seen in Figure 1.8, we see that the higher intercept and larger negative gradient mean that at some point, the two lines will intercept. Using our models we can predict the first Olympic games when the women's winning time will be faster than the men's. According to our models this will be in the 2592 Olympics (the actual answer has been rounded up to the nearest Olympic year and has been computed in Matlab using the exact data so you might find slight differences due to rounding.) (see Exercise EX 1.8).

As with the point predictions for individual models, we should not place too much confidence on this prediction coming about. Not only is the prediction incredibly precise, it is also a very long time from our last observed data point. Can we assume that the relationship between winning time and Olympic year will continue this far into the future? To assume that it can is also to assume that there will, eventually, be a winning time of 0 seconds and we know that this is impossible.

1.2.2 Summary

In the previous sections we have seen how we can fit a simple linear model to a small dataset and use the resulting model to make predictions. We have also described some of the limitations of making predictions in this way and we will

n	x_n	t_n	$x_n t_n$	x_n^2
1	1928	12.20	23521.6	3.7172×10^6
2	1932	11.90	22990.8	3.7326×10^6
3	1936	11.50	22264.0	3.7481×10^6
4	1948	11.90	23181.2	3.7947×10^6
5	1952	11.50	22448.0	3.8103×10^6
6	1956	11.50	22494.0	3.8259×10^6
7	1960	11.00	21560.0	3.8416×10^6
8	1964	11.40	22389.6	3.8573×10^6
9	1968	11.00	21648.0	3.8730×10^6
10	1972	11.07	21830.0	3.8888×10^6
11	1976	11.08	21894.1	3.9046×10^6
12	1980	11.06	21898.8	3.9204×10^6
13	1984	10.97	21764.5	3.9363×10^6
14	1988	10.54	20953.5	3.9521×10^6
15	1992	10.82	21553.4	3.9681×10^6
16	1996	10.94	21836.2	3.9840×10^6
17	2000	11.12	22240.0	4.0000×10^6
18	2004	10.93	21903.7	4.0160×10^6
19	2008	10.78	21646.2	4.0321×10^6
$(1/N)\sum_{n=1}^{N}$	1970.74	11.22	22106.2	3.8844×10^6

TABLE 1.3: Olympic women's 100 m data.

introduce alternative techniques that overcome these limitations in later Chapters. Up to this point, our attributes (x_n) have been individual numbers. We will now see how the linear model can be extended to larger sets of attributes, enabling us to model more complex relationships.

1.3 Vector/matrix notation

In many applications, we will be interested in problems where each data-point is described by a set of several attributes. For example, we might decide that using only the Olympic year is unsuitable for model a model of Olympic sprint data. A model that used the Olympic year and each athlete's personal best might be more accurate. Using $s_1, s_2, \ldots, s_8$ to denote the personal best times for the athletes running in lanes 1 to 8, a possible linear model might consist of:

$$\begin{aligned} t = f(x, s_1, \ldots, s_5; w_0, \ldots, w_6) \quad &= \quad w_0 + w_1 x + w_2 s_1 + w_3 s_2 + w_4 s_3 \\ &\quad\; + w_5 s_4 + w_6 s_5 + w_7 s_6 + w_8 s_7 + w_9 s_8. \end{aligned}$$

We could go through the analysis of the previous sections to find $\widehat{w_0}, \ldots, \widehat{w_9}$. After taking partial derivatives of the loss function, we would be left with 10 equations

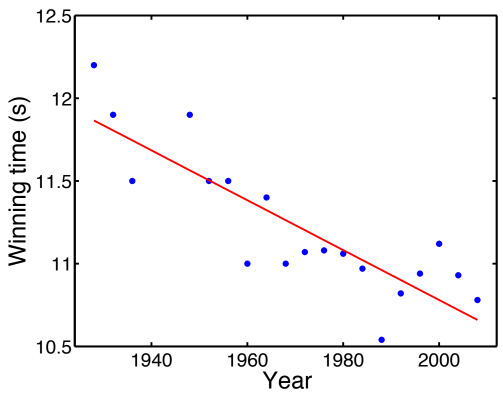


FIGURE 1.7: Women's Olympic 100 m data with a linear model that minimises the squared loss.

that would need to be re-arranged and substituted into one another. This would be a time consuming exercise and would rapidly become infeasible as the number of variables we wanted to include increased further – Machine Learning applications with thousands of variables are not uncommon. Fortunately there is an alternative – using vectors and matrices.

As this is an area that some readers will find unfamiliar, we shall now devote some time to describing vector and matrix notation and how to perform mathematical operations with quantities in vector and matrix form. Readers familiar with these concepts could jump straight to Section 1.4.

Comment 1.3 – Scalars, vectors and matrices: We will follow the standard convention of representing scalar values by letters (e.g. x), vectors by bold lowercase letters (e.g. $\mathbf{x}$) and matrices by bold uppercase letters (e.g. $\mathbf{X}$). Whilst we shall consistently stick to this notation, different communities have different ways of defining vectors. For example, $\overline{x}$ is common for a vector x.

The 9 attributes for each data point (8 personal bests and Olympic year) can be combined into a single variable by stacking them together to form a vector. We will denote vectors with bold lower case letters, e.g. $\mathbf{x}_n$ (see Comment 1.3). Often we will need to refer to individual elements within a particular vector or matrix and will use *indices* to make it clear which element we're referring to. For example, the first element of the vector $\mathbf{x}_n$ would be denoted x_{n1}, the ith by x_{ni}.

If we want to show all of the elements in a vector, we write it out in a tabular fashion, surrounded by square brackets. Here are examples of vectors of length 2 and 4:

$$\mathbf{x}_n = \begin{bmatrix} x_{n1} \\ x_{n2} \end{bmatrix}, \; \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}.$$

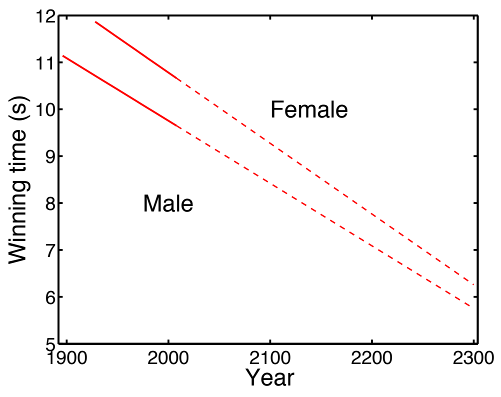


FIGURE 1.8: Male and female functions extrapolated into the future.

Comment 1.4 – Vector transpose: The transpose of a vector $\mathbf{x}$, denoted $\mathbf{x}^{\mathsf{T}}$, is obtained by rotating the vector such that rather than having one column and several rows, it has one row and several columns. For example:

$$\mathbf{x} = \begin{bmatrix} 4 \\ 7 \\ 11 \\ -2 \end{bmatrix}, \quad \mathbf{x}^{\mathsf{T}} = [4, 7, 11, -2].$$

It is often a bit clumsy to keep drawing vectors as columns so we will often draw them as rows, and use the transpose operator (see Comment 1.4) to show that they should be rotated. If we assume that we have D attributes, we would define $\mathbf{x}_n$ as $\mathbf{x}_n = [x_{n1}, \ldots, x_{nD}]^{\mathsf{T}}$. In the case of our Olympic data, $\mathbf{x} = [\text{Year}, s_1, s_2, \ldots, s_8]^{\mathsf{T}}$.

Comment 1.5 – Matrix/vector dimensions and indexing: If we are quoting the size (or the dimension) of a matrix or vector, we give two numbers, starting with the number of rows. For example,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$$

has dimension 3×2. A vector is a special case of a matrix where the second dimension is 1. For example,

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

could be thought of as a matrix with dimension 4×1.
When indexing elements within a vector, a single number is sufficient (e.g. y_3 for the third element in $\mathbf{y}$ above. When indexing a matrix, we will use two subscripts, starting with the row. For example, a_{21} represents the item in the second row and first column of $\mathbf{A}$ (above). Note that sometimes we will also have a subscript denoting the object index. For example $\mathbf{x}_n$ is the vector holding the nth set of attributes. This index, if present, will always come first. It should be obvious from the context whether or not this index is present.

Before we embark on adding additional variables, it is worthwhile to repeat the analysis of the original model ($t = w_0 + w_1 x$) in vector form. This will allow us to compare the expressions we obtain for $\widehat{w_0}$ and $\widehat{w_1}$ in both cases. The first step is to combine w_0 and w_1 into a single parameter vector $\mathbf{w}$ and create data vectors $\mathbf{x}_n$ by augmenting each x_n with a 1, i.e.

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \quad \mathbf{x}_n = \begin{bmatrix} 1 \\ x_n \end{bmatrix}$$

The model can be expressed in terms of $\mathbf{x}_n$ and $\mathbf{w}$ as (matrix/vector multiplication is defined in Comment 1.7):

$$f(x_n; w_0, w_1) = \mathbf{w}^\mathsf{T}\mathbf{x}_n = w_0 + w_1 x_n.$$

We can replace any instance of $w_0 + w_1 x$ by $\mathbf{w}^\mathsf{T}\mathbf{x}$. For example, our squared loss $\mathcal{L}$ can be expressed as

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}(t_n - \mathbf{w}^\mathsf{T}\mathbf{x}_n)^2. \tag{1.12}$$

In actual fact, we can express this average loss as the following function of various vectors and matrices which will be easier to manipulate:

$$\mathcal{L} = \frac{1}{N}(\mathbf{t} - \mathbf{X}\mathbf{w})^\mathsf{T}(\mathbf{t} - \mathbf{X}\mathbf{w}).$$

To see how this is equivalent to Equation 1.12, we start by combining all $\mathbf{x}_n$ into one matrix $\mathbf{X}$, and all t_n into one vector $\mathbf{t}$:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^{\mathsf{T}} \\ \mathbf{x}_2^{\mathsf{T}} \\ \vdots \\ \mathbf{x}_N^{\mathsf{T}} \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}.$$

Comment 1.6 – Matrix transpose: For a matrix, $\mathbf{X}$, the transpose, $\mathbf{X}^{\mathsf{T}}$ is formed by turning each row into a column and each column into a row. For example, if $\mathbf{Y} = \mathbf{X}^{\mathsf{T}}$, then $Y_{ij} = X_{ji}$.

$$\mathbf{X} = \begin{bmatrix} 1 & 4 \\ 3 & 6 \\ -2 & 11 \end{bmatrix}, \quad \mathbf{X}^{\mathsf{T}} = \begin{bmatrix} 1 & 3 & -2 \\ 4 & 6 & 11 \end{bmatrix}.$$

Comment 1.7 – Matrix multiplication: To proceed, we must introduce the concept of matrix multiplication. Taking the product, $\mathbf{AB}$ of an $N \times M$ matrix $\mathbf{A}$ and a $P \times Q$ matrix $\mathbf{B}$ is only possible if $M = P$, i.e. the number of columns in $\mathbf{A}$ is equal to the number of rows in $\mathbf{B}$. Assuming that this is the case, the product, $\mathbf{C} = \mathbf{AB}$ is the $N \times Q$ matrix defined such that

$$C_{ij} = \sum_k A_{ik} B_{kj}.$$

It is often helpful to draw the matrices, for example

$$\begin{array}{cc} & \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} \\ \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} & \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} & a_{11}b_{13} + a_{12}b_{23} \\ a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} & a_{11}b_{13} + a_{12}b_{23} \end{bmatrix} \end{array}$$

where we can think of computing elements of $\mathbf{C}$ by working simultaneously across the relevant row of $\mathbf{A}$ and column of $\mathbf{B}$.
A special case that we will meet regularly is the *inner product* between two column vectors, defined as $z = \mathbf{x}^{\mathsf{T}}\mathbf{y}$, the result of which is a scalar. Both vectors must be of the same length and the transpose ensures that the number of columns in $\mathbf{x}$ is the same as the number of rows in $\mathbf{y}$. Applying the same technique as that for matrices, we see that

$$z = \sum_k x_k y_k.$$

Therefore, if we perform the matrix multiplication $\mathbf{Xw}$ we will end up with a vector which looks like this:

$$\mathbf{Xw} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix} \times \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} w_0 + w_1 x_1 \\ w_0 + w_1 x_2 \\ \vdots \\ w_0 + w_1 x_N \end{bmatrix}.$$

Subtracting this from $\mathbf{t}$ will give us:

$$\mathbf{t} - \mathbf{X}\mathbf{w} = \left[\begin{array}{c} t_1 - w_0 - w_1 x_1 \\ t_2 - w_0 - w_1 x_2 \\ \vdots \\ t_N - w_0 - w_1 x_N \end{array}\right].$$

and we can use a single multiplication and transpose to neatly perform the squaring and summation and obtain our original loss function:

$$\begin{aligned}(\mathbf{X}\mathbf{w} - \mathbf{t})^\mathsf{T}(\mathbf{X}\mathbf{w} - \mathbf{t}) &= (w_0 + w_1x_1 - t_1)^2 + (w_0 + w_1x_2 - t_2)^2 + \cdots + (w_0 + w_1x_N - t_N)^2 \\ &= \sum_{n=1}^{N}(w_0 + w_1x_n - t_n)^2 \\ &= \sum_{n=1}^{N}(t_n - f(x_n; w_0, w_1))^2.\end{aligned}$$

Therefore, our loss can be written compactly as:

$$\mathcal{L} = \frac{1}{N}(\mathbf{t} - \mathbf{X}\mathbf{w})^\mathsf{T}(\mathbf{t} - \mathbf{X}\mathbf{w}), \qquad (1.13)$$

and the following loss expressions are all equivalent:

$$\mathcal{L} = \frac{1}{N}(\mathbf{t} - \mathbf{X}\mathbf{w})^\mathsf{T}(\mathbf{t} - \mathbf{X}\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(t_n - \mathbf{w}^\mathsf{T}\mathbf{x}_n)^2 = \frac{1}{N}\sum_{n=1}^{N}(t_n - (w_0 + w_1x_n))^2$$

Comment 1.8 – Transpose of a product: The transpose of a matrix product, $(\mathbf{X}\mathbf{w})^\mathsf{T}$ can be expanded by reversing the order of multiplication and transposing the two individual matrices

$$(\mathbf{X}\mathbf{w})^\mathsf{T} = \mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}.$$

To deal with more complex forms, we can apply the same result several times. For example

$$\begin{aligned}(\mathbf{ABCD})^\mathsf{T} &= ((\mathbf{AB})(\mathbf{CD}))^\mathsf{T} \\ &= (\mathbf{CD})^\mathsf{T}(\mathbf{AB})^\mathsf{T} \\ &= \mathbf{D}^\mathsf{T}\mathbf{C}^\mathsf{T}\mathbf{B}^\mathsf{T}\mathbf{A}^\mathsf{T}\end{aligned}$$

It will be easier to work with this matrix loss once we have multiplied out the brackets. Noting that order is important in matrix multiplication (this is implied by the restriction on sizes discusses in Comment 1.7) and the definition for the transpose of a product given in Comment 1.8:

$$\begin{aligned}\mathcal{L} &= \frac{1}{N}(\mathbf{X}\mathbf{w} - \mathbf{t})^\mathsf{T}(\mathbf{X}\mathbf{w} - \mathbf{t}) \\ &= ((\mathbf{X}\mathbf{w})^\mathsf{T} - \mathbf{t}^\mathsf{T})(\mathbf{X}\mathbf{w} - \mathbf{t}) \\ &= \frac{1}{N}(\mathbf{X}\mathbf{w})^\mathsf{T}\mathbf{X}\mathbf{w} - \frac{1}{N}\mathbf{t}^\mathsf{T}\mathbf{X}\mathbf{w} - \frac{1}{N}(\mathbf{X}\mathbf{w})^\mathsf{T}\mathbf{t} + \frac{1}{N}\mathbf{t}^\mathsf{T}\mathbf{t} \\ &= \frac{1}{N}\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w} - \frac{2}{N}\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{t} + \frac{1}{N}\mathbf{t}^\mathsf{T}\mathbf{t}. \qquad (1.14)\end{aligned}$$

The two terms $\mathbf{t}^\mathsf{T}\mathbf{X}\mathbf{w}$ and $\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{t}$ are the transpose of one another (using the identity for the transpose of a product) and also scalars (satisfy yourself that the result is a 1×1 matrix and hence a scalar). This implies that they must be the same and can therefore be combined.

Differentiating loss in vector/matrix form: We now require the value of the vector $\mathbf{w}$ corresponding to a turning point (minimum) of $\mathcal{L}$. To do this, we must take the partial derivate of $\mathcal{L}$ with respect to the *vector* $\mathbf{w}$. This involves taking partial derivatives of $\mathcal{L}$ with respect to each element of $\mathbf{w}$ in turn and then stacking the results into a vector. It is worth explicitly doing this in this instance although we will see later that we can actually obtain $\frac{\partial \mathcal{L}}{\partial \mathbf{w}}$ directly in vector form. In our two variable case, this vector is

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial w_0} \\ \frac{\partial \mathcal{L}}{\partial w_1} \end{bmatrix},$$

the vector containing the partial derivatives of $\mathcal{L}$ with respect to w_0 and w_1. The two elements of this vector should be the same as Equations 1.7 and 1.6 respectively. We can check that our loss is indeed correct by manually differentiating Equation 1.13 with respect to the two parameters. Firstly, we need the multiplied out expression

$$\mathcal{L} = \frac{1}{N}(\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w} - 2\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{t} + \mathbf{t}^\mathsf{T}\mathbf{t}).$$

The last term doesn't include either w_0 or w_1 so we can ignore it. When multiplied out, the first term is (see Exercise EX 1.3)

$$w_0^2 \frac{1}{N}\left(\sum_{n=1}^N X_{n0}^2\right) + 2w_0w_1\frac{1}{N}\left(\sum_{n=1}^N X_{n0}X_{n1}\right) + w_1^2\frac{1}{N}\left(\sum_{n=1}^N X_{n1}^2\right),$$

where X_{n0} is the first element of the nth row of $\mathbf{X}$, i.e. the first element of the nth data object and X_{n1} is the second (we've started numbering from zero to maintain the relationship with w_0). Similarly, the second term is equivalent to

$$2w_0\frac{1}{N}\left(\sum_{n=1}^N X_{n0}t_n\right) + 2w_1\frac{1}{N}\left(\sum_{n=1}^N X_{n1}t_n\right).$$

Combining these and noting that, in our previous notation, $X_{n0} = 1$ and $X_{n1} = x_n$ results in

$$w_0^2 + 2w_0w_1\frac{1}{N}\left(\sum_{n=1}^N x_n\right) + w_1^2\frac{1}{N}\left(\sum_{n=1}^N x_{n1}^2\right) - 2w_0\frac{1}{N}\left(\sum_{n=1}^N t_n\right) - 2w_1\frac{1}{N}\left(\sum_{n=1}^N x_nt_n\right).$$

Recalling our shorthand for the various averages and differentiating with respect to w_0 and w_1 results in

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_0} &= 2w_0 + 2w_1\bar{x} - 2\bar{t} \\ \frac{\partial \mathcal{L}}{\partial w_1} &= 2w_0\bar{x} + 2w_1\overline{x^2} - 2\overline{xt}. \end{aligned}$$

It is left as an informal exercise to show that these are indeed equivalent to the derivates obtained from the non-vectorised loss function (Equations 1.7 and 1.6).

Fortunately, there are many standard identities that we can use that enable us

$f(\mathbf{w})$	$\frac{\partial f}{\partial \mathbf{w}}$
$\mathbf{w}^\mathsf{T}\mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^\mathsf{T}\mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^\mathsf{T}\mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^\mathsf{T}\mathbf{C}\mathbf{w}$	$2\mathbf{C}\mathbf{w}$

TABLE 1.4: Some useful identities when differentiating with respect to a vector.

to differentiate the vectorised expression directly. Those that we will need are shown in Table 1.4.

From these identities, and equating the derivative to zero, we can directly obtain the following:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \frac{2}{N}\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w} - \frac{2}{N}\mathbf{X}^\mathsf{T}\mathbf{t} = 0 \\
\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w} &= \mathbf{X}^\mathsf{T}\mathbf{t}. \qquad (1.15)
\end{aligned}$$

Comment 1.9 – Identity matrix: we will regularly come across the identity matrix $\mathbf{I}_N$. It is the $N \times N$ matrix with ones on the diagonal and zeros elsewhere.

$$\mathbf{I}_1 = 1, \quad \mathbf{I}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Often, the size of the identity matrix will be obvious from the expression it's found in. In these cases, we will omit the size subscript.
A key property of the identity matrix is that any vector or matrix multiplied by a suitably sized identity matrix is equal to the original matrix or vector. For example, if $\mathbf{y} = [y_1, \ldots, y_D]^\mathsf{T}$ and $\mathbf{I}_D$ is the $D \times D$ identity matrix,

$$\mathbf{y}^\mathsf{T}\mathbf{I}_D = \mathbf{y}, \ \mathbf{I}\mathbf{y} = \mathbf{y}.$$

Similarly, for an $N \times M$ matrix,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1M} \\ a_{21} & a_{22} & \ldots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \ldots & a_{NM} \end{bmatrix}$$

$$\mathbf{A}\mathbf{I}_M = \mathbf{A}, \ \mathbf{I}_N\mathbf{A} = \mathbf{A}.$$

Multiplying a scalar by an identity results in a matrix with the scalar value on each diagonal element. An example that crops up a lot is:

$$\sigma^2\mathbf{I}_M = \begin{bmatrix} \sigma^2 & 0 & \ldots & 0 \\ 0 & \sigma^2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma_2 \end{bmatrix}$$

Comment 1.10 – Matrix inverse: The inverse of a matrix $\mathbf{A}$ is defined as the matrix, $\mathbf{A}^{-1}$ that satisfies $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. We don't provide the general form for inverting a matrix here, but from school mathematics, a 2×2 matrix can be inverted with the following formula:

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad \mathbf{A}^{-1} = \frac{1}{ad - bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

A special case that we will come across regularly is the inverse of a matrix that only has values on the diagonal (i.e. all off-diagonal elements are zero). The inverse of such a matrix is another diagonal matrix where each diagonal element is simply the inverse of the corresponding element in the original. For example,

$$\mathbf{A} = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{DD} \end{bmatrix}, \mathbf{A}^{-1} = \begin{bmatrix} a_{11}^{-1} & 0 & \dots & 0 \\ 0 & a_{22}^{-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{DD}^{-1} \end{bmatrix}$$

It is worth noting that this definition implies that the inverse of an identity matrix (see Comment 1.9) is simply another identity matrix:

$$\mathbf{I}^{-1} = \mathbf{I}.$$

The final step in deriving an expression $\widehat{\mathbf{w}}$, the optimum value of $\mathbf{w}$ is re-arranging Equation 1.15. We cannot divide both sides by $\mathbf{X}^\mathsf{T}\mathbf{X}$ (division isn't defined for matrices) but we can pre-multiply both sides by a matrix that will cancel the $\mathbf{X}^\mathsf{T}\mathbf{X}$ from the left (leaving only an identity matrix, see Comment 1.9). This matrix is called the matrix inverse of $\mathbf{X}^\mathsf{T}\mathbf{X}$ (see Comment 1.10) and is denoted by $(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}$. Pre-multiplying both sides of (1.15) with $(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}$, we obtain:

$$\mathbf{Iw} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{t}.$$

As $\mathbf{Iw} = \mathbf{w}$ (from the definition of the identity matrix), we are left with a matrix equation for $\widehat{\mathbf{w}}$, the value of $\mathbf{w}$ that minimises the loss:

$$\widehat{\mathbf{w}} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{t} \tag{1.16}$$

1.3.1 Example

We can check that our matrix equation is doing exactly the same as the scalar equations we got previously by multiplying it out. In two dimensions,

$$\mathbf{X}^\mathsf{T}\mathbf{X} = \begin{bmatrix} \sum_{n=1}^N x_{n0}^2 & \sum_{n=1}^N x_{n0}x_{n1} \\ \sum_{n=1}^N x_{n1}x_{n0} & \sum_{n=1}^N x_{n1}^2 \end{bmatrix}.$$

Using $\bar{x}$ to denote averages, this can be re-written as

$$\mathbf{X}^\mathsf{T}\mathbf{X} = N\begin{bmatrix} \overline{x_0^2} & \overline{x_0x_1} \\ \overline{x_1x_0} & \overline{x_1^2} \end{bmatrix}.$$

The identity for the inverse of a 2×2 matrix (see Comment 1.10) enables us to invert this

$$(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1} = \frac{1}{N}\frac{1}{\overline{x_0^2}\;\overline{x_1^2} - \overline{x_1 x_0}\;\overline{x_0 x_1}} \begin{bmatrix} \overline{x_1^2} & -\overline{x_0 x_1} \\ -\overline{x_1 x_0} & \overline{x_0^2} \end{bmatrix}.$$

We need to multiply this by $\mathbf{X}^\mathsf{T}\mathbf{t}$ which is (jumping straight to the average notation):

$$N \begin{bmatrix} \overline{x_0 t} \\ \overline{x_1 t} \end{bmatrix}.$$

Now, we know that x_{n0} is 1 always and redefining x_{n1} as x_n (to be consistent with the scalar notation), we need to evaluate:

$$\widehat{\mathbf{w}} = \frac{1}{N}\frac{1}{\overline{x^2} - \overline{x}\;\overline{x}} \begin{bmatrix} \overline{x^2} & -\overline{x} \\ -\overline{x} & 1 \end{bmatrix} \times N \begin{bmatrix} \overline{t} \\ \overline{xt} \end{bmatrix},$$

which is:

$$\widehat{\mathbf{w}} = \begin{bmatrix} \widehat{w_0} \\ \widehat{w_1} \end{bmatrix} = \left(\frac{1}{\overline{x^2} - \overline{x}\;\overline{x}}\right) \begin{bmatrix} \overline{x^2}\;\overline{t} - \overline{x}\;\overline{xt} \\ -\overline{x}\;\overline{t} + \overline{xt} \end{bmatrix}. \tag{1.17}$$

Starting with $\widehat{w}_1$ (the second line)

$$\widehat{w}_1 = \frac{\overline{xt} - \overline{x}\;\overline{t}}{\overline{x^2} - \overline{x}\;\overline{x}},$$

exactly as before. $\widehat{w_0}$ requires a little more rearrangement and it is easier to work backwards. Starting from our original expression and substituting this new expression for $\widehat{w_1}$

$$\begin{aligned}
\widehat{w}_0 &= \overline{t} - \widehat{w_1}\overline{x} \\
&= \overline{t} - \overline{x}\frac{\overline{xt} - \overline{x}\;\overline{t}}{\overline{x^2} - \overline{x}\;\overline{x}} \\
&= \overline{t}\left(\frac{\overline{x^2} - \overline{x}\;\overline{x}}{\overline{x^2} - \overline{x}\;\overline{x}}\right) - \overline{x}\frac{\overline{xt} - \overline{x}\;\overline{t}}{\overline{x^2} - \overline{x}\;\overline{x}} \\
&= \frac{\overline{t}\;\overline{x^2} - \overline{t}\;\overline{x}\;\overline{x} - \overline{x}\;\overline{xt} + \overline{x}\;\overline{x}\;\overline{t}}{\overline{x^2} - \overline{x}\;\overline{x}} \\
&= \frac{\overline{t}\overline{x^2} - \overline{x}\;\overline{xt}}{\overline{x^2} - \overline{x}\;\overline{x}},
\end{aligned}$$

which is exactly the first line in Equation 1.17 as required.

1.3.2 Numerical example

To help those readers who are not familiar with working with vectors and matrices, we will now repeat the synthetic linear regression example we saw in the previous section. The data, in matrix notation is:

$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 1 & 5 \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} 4.8 \\ 11.3 \\ 17.2 \end{bmatrix}.$$

Examining Equation 1.16 we see that the first quantity we need to calculate is $\mathbf{X}^\mathsf{T}\mathbf{X}$

$$\mathbf{X}^\mathsf{T}\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 3 & 9 \\ 9 & 35 \end{bmatrix}.$$

Using the formula provided above, we compute the inverse as

$$(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1} = \frac{1}{24}\begin{bmatrix} 35 & -9 \\ -9 & 3 \end{bmatrix}.$$

Multiplying by $\mathbf{X}^\mathsf{T}$,

$$(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T} = \frac{1}{24}\begin{bmatrix} 35 & -9 \\ -9 & 3 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3 & 5 \end{bmatrix} = \frac{1}{24}\begin{bmatrix} 26 & 8 & -10 \\ -6 & 0 & 6 \end{bmatrix}.$$

Finally, we multiply this matrix by $\mathbf{t}$:

$$\left((\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\right)\mathbf{t} = \frac{1}{24}\begin{bmatrix} 26 & 8 & -10 \\ -6 & 0 & 6 \end{bmatrix} \times \begin{bmatrix} 4.8 \\ 11.3 \\ 17.2 \end{bmatrix} = \begin{bmatrix} 1.8 \\ 3.1 \end{bmatrix}.$$

Therefore, our formula is $f(x; w_0, w_1) = 1.8 + 3.1x$, exactly as before.

1.3.3 Making predictions

Given a new vector of attributes x_{new}, the prediction from the model, t_{new}, is computed as:

$$t_{\mathsf{new}} = \widehat{\mathbf{w}}^\mathsf{T}\mathbf{x}_{\mathsf{new}}.$$

1.3.4 Summary

In the previous sections, we have described our linear model in terms of vectors and matrices. The result is a very useful model – our expression for $\widehat{\mathbf{w}}$ makes no assumptions as to the number of parameters included in $\widehat{\mathbf{w}}$ (its length). We can therefore compute $\widehat{\mathbf{w}}$ and make predictions for any linear model of the form:

$$t_n = w_1 x_{n1} + w_2 x_{n2} + w_3 x_{n3} + \ldots$$

This is a powerful tool – many real datasets are described by more than one attribute and for many, a linear model of this type will be appropriate. We have also learnt that predictions from this model are very precise and that this is not always sensible. We shall see how to overcome this in later chapters.

The attributes that make up $\mathbf{x}_n$ could be measurements of different properties (e.g. winning times and personal bests). Alternatively, they could be the result of applying a set of functions to an individual attribute like the Olympic year: x_n. This allows us to extend our armoury beyond models corresponding to straight lines and is the subject of the next section.

1.4 Nonlinear response from a linear model

At the start of this chapter, we made the assumption that we could model the relationship between time and Olympic 100 m sprint times using a linear function. In many real applications, this is too restrictive. Even for the 100 m data it could be argued that it is far too simplistic – the linear model predicts that in the year 3000, the time will be -3.5 seconds! Fortunately, we can use exactly the same framework we have already described to fit a family of more complex models through a transformation of the attributes.

The linear model we have seen thus far

$$f(x; \mathbf{w}) = w_0 + w_1 x$$

is linear in both the parameters (w) and the data (x) (see Comment 1.1). The linearity in the parameters is desirable from a computational point of view as the solution that minimises the squared loss can be found exactly via Equations 1.8 and 1.10. Consider augmenting our data matrix $\mathbf{X}$ with an additional column, x_n^2:

$$\mathbf{x}_n = \begin{bmatrix} 1 \\ x_n \\ x_n^2 \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 \end{bmatrix}$$

and adding an extra parameter to $\mathbf{w}$:

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix},$$

resulting in:

$$f(x; \mathbf{w}) = \mathbf{w}^\mathsf{T}\mathbf{x} = w_0 + w_1 x + w_2 x^2.$$

As the model is still linear in the parameters, we can use Equation 1.16 to find $\mathbf{w}$ but the function we are fitting is **quadratic** in the data. Figure 1.9 shows an example of using exactly this method to fit a function quadratic in the data to a suitable dataset (solid line) (Matlab script: `synthquad.m`). Also shown is the function we get if we try and fit our original linear (in the data) model (dashed line, $t = w_0 + w_1 x$). It is clear from the quality of the fit to the data that the quadratic model is a more appropriate model for this data.

More generally, we can add as many powers of x as we like to get a **polynomial** function of any order. For a Kth order polynomial, our augmented data matrix will be:

$$\mathbf{X} = \begin{bmatrix} x_1^0 & x_1^1 & x_1^2 & \cdots & x_1^K \\ x_2^0 & x_2^1 & x_2^2 & \cdots & x_2^K \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ x_N^0 & x_N^1 & x_N^2 & \cdots & x_N^K \end{bmatrix} \tag{1.18}$$

(where $x^0 = 1$) and our function can be written in the more general form:

$$f(x; \mathbf{w}) = \sum_{k=0}^{K} w_k x^k.$$

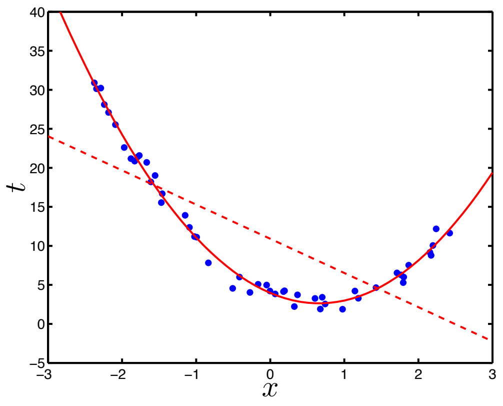


FIGURE 1.9: Example of linear (dashed) and quadratic (solid) models fitted to dataset generated from a quadratic function.

Figure 1.10 shows the effect of fitting an 8th order polynomial function to the 100 m sprint data that we have seen previously (Matlab script: `olymppoly.m`). Comparing with Figures 1.5 and 1.6, does the 8th order model look better than the 1st order model? To answer this question, we need to be more precise about what we mean by better. For models built to make predictions, the best model is arguably the one that produces the best predictions. We shall return to the issue of model selection in more detail in Section 1.5. However, two things are immediately apparent and warrant description. Firstly, the 8th order polynomial gets closer to the observed data than the 1st order polynomial (original model). This is reflected in a lower value of the loss function: $\mathcal{L}^8 = 0.459$, $\mathcal{L}^1 = 1.358$ (where $\mathcal{L}^k$ is the loss achieved with a kth order polynomial). In fact, increasing the polynomial order will always result in a model that gets closer to the training data. Secondly, the predictions (shown by the dashed line) do not look sensible, particularly outside the range of the observed data.

We are not restricted to polynomial functions. We are free to define any set of K functions of x, $h_k(x)$:

$$\mathbf{X} = \begin{bmatrix} h_1(x_1) & h_2(x_1) & \cdots & h_K(x_1) \\ h_1(x_2) & h_2(x_2) & \cdots & h_K(x_2) \\ \vdots & \vdots & \ldots & \vdots \\ h_1(x_N) & h_2(x_N) & \cdots & h_K(x_N) \end{bmatrix}$$

which can be anything that we feel may be appropriate for the data available. For example, there appears to be a slight periodic trend in the 100 m data. A suitable

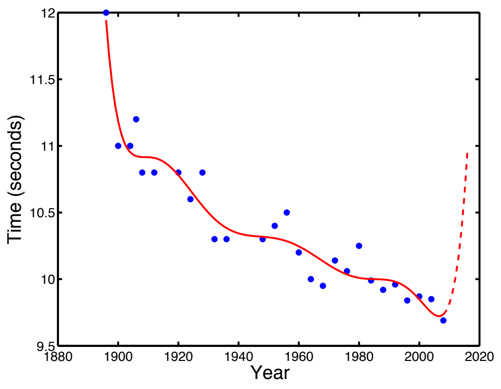


FIGURE 1.10: 8th order polynomial fitted to the Olympic 100 m men's sprint data.

set of functions might be:

$$
\begin{aligned}
h_1(x) &= 1\\
h_2(x) &= x\\
h_3(x) &= \sin\left(\frac{x-a}{b}\right)\\
f(x;\mathbf{w}) &= w_0 + w_1 x + w_2 \sin\left(\frac{x-a}{b}\right).
\end{aligned}
$$

This model has 5 parameters – w_0, w_1, w_2, a, b. Unfortunately, only the first three can be inferred using the procedures that we have developed. The last two, a and b, appear inside a nonlinear (sine) function. As such, taking partial derivatives with respect to these parameters and equating to zero will not result in a set of equations that can be solved analytically. There are many ways of overcoming this problem, the simplest being a search over all values of a and b in some sensible range. However, we will ignore this problem for now and assume that we know of suitable values. If a and b are fixed, we can set the remaining parameters (w_0, w_1, w_2) using the expressions we derived previously. Assuming a and b are fixed ($a = 2660, b = 4.3$), Figure 1.11 shows a least squares fit using this model. In this case $\mathcal{L} = 1.1037$ so it is fitting the observed data better than the first order polynomial but not as well as the 8th order polynomial. The various model components are clearly visible in Figure 1.11: the constant term ($w_0 = 36.610$), the downward linear trend ($w_1 = -0.013$) and the nonlinear sinusoidal term ($w_3 = -0.133$) causing oscillations. Notice also how the values for w_0 and w_1 are very similar to those for the first order polynomial model (c.f. Figure 1.5) – we have added an oscillating component around our original linear model.

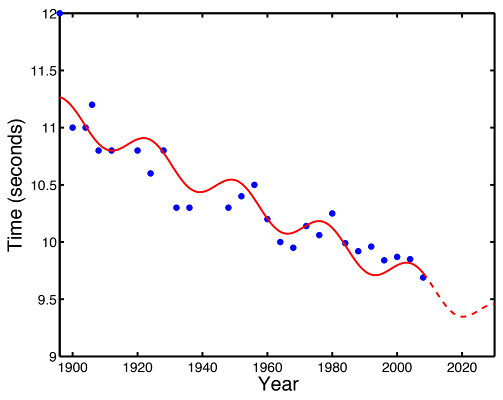


FIGURE 1.11: Least squares fit of $f(x; \mathbf{w}) = w_0 + w_1 x + w_2 \sin\left(\frac{x-a}{b}\right)$ to the 100 m sprint data ($a = 2660, b = 4.3$).

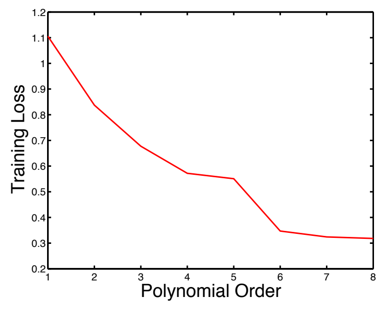
1.5 Generalisation and over-fitting

In Section 1.4, we posed the question of which was better, the 1st or 8th order polynomial. Given that our original aim in building these models was to make predictions, it makes sense that the best model is the one which is able to make the most accurate predictions. Such a model will be one that can *generalise* beyond the examples we have for *training* (our Olympic data up to 2008 for example). Ideally, we would like to choose the model that performs best (i.e. minimises the loss) on this unseen data but, by the very nature of the problem, this data is unavailable.

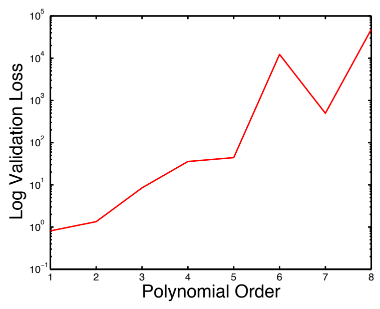
Figure 1.10 gave an early indication that we should be very suspicious of using the loss on the training data to choose a model that will be used to make predictions. The plot shows an 8th order polynomial fit to the men's 100 m data which has a much lower loss on the training data than a first order polynomial. At the same time, the predictions for future Olympics are very poor. For this data, a model based on an 8th order polynomial pays too much attention to the training data (it *over-fits*) and as a result does not generalise well to new data. As we make models more and more complex, they will be able to get closer and closer to the data that we have already seen. Unfortunately, beyond a certain point, the quality of the predictions can deteriorate rapidly. Determining the optimal model complexity such that it is able to generalise well without **over-fitting** is very challenging. This tradeoff is often referred to as the bias-variance tradeoff and we will briefly mention this in Section 2.9.

1.5.1 Validation data

One common way to overcome this problem is to use a second dataset, often referred to as a **validation** set. It is so called as it is used to validate the predictive performance of our model. The validation data could be provided separately or we could create it by removing some data from the original *training* set. For example, in our 100 m data, we could remove all Olympics since 1980 from the training set and make these the validation set. To choose between a set of models, we train each one on the reduced training set and then compute their loss on the validation set. Plots of the training and (log) validation losses can be seen in Figure 1.12(a) and Figure 1.12(b) respectively. The training loss decreases **monotonically** as the polynomial order (and hence **model complexity**) increases. However, the validation loss increases rapidly as the polynomial order increases suggesting that a first order polynomial has best **generalisation** ability and will produce the most reliable predictions. This hypothesis is easily tested. In Figure 1.13 we can see the data (labeled as training and validation) and 1st, 4th and 8th order polynomial functions (Matlab script: `olympval.m`). It is clear to see that for this data, that had we been performing this task in 1979, a first order model would indeed have given the best predictions.



(a) Training loss for the Olympic men's 100 m data.



(b) Log validation loss for the Olympic men's 100 m data. When using the squared loss, this is also known as the squared predictive error and measured how close the predicted values are to the true values. Note that the log loss is plotted as the value increases so rapidly.

FIGURE 1.12: Training and validation loss for Olympic men's 100 m data.

1.5.2 Cross-validation

The loss that we calculate from validation data will be sensitive to the choice of data in our validation set. This is particularly problematic if our dataset (and hence our validation set) is small. **Cross-validation** is a technique that allows us to make more efficient use of the data we have.

K-fold cross-validation splits the data into K equally (or as close to equal as

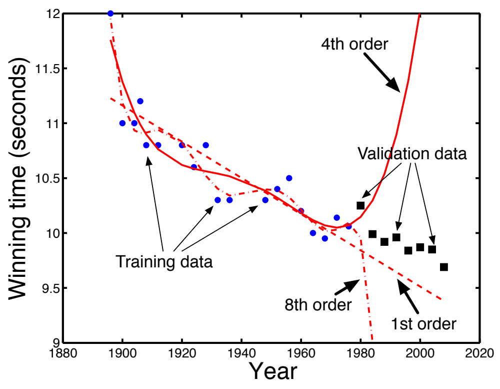


FIGURE 1.13: Generalisation ability of first, fourth and eighth order polynomials on Olympic men's 100 m data.

possible) sized blocks, illustrated in Figure 1.14. Each block takes its turn as a validation set for a training set comprised of the other $K-1$ blocks. Averaging over the resulting K loss values gives us our final loss value. An extreme case of K-fold cross-validation is where $K = N$, the number of observations in our dataset: each data observation is held out in turn and used to test a model trained on the other $N-1$ objects. This particular form of cross-validation is given the name Leave-One-Out Cross Validation (LOOCV). The average squared validation loss for LOOCV is:

$$\mathcal{L}^{CV} = \frac{1}{N}\sum_{n=1}^{N}(t_n - \widehat{\mathbf{w}}_{-n}^{\mathsf{T}}\mathbf{x}_n)^2, \tag{1.19}$$

where $\widehat{\mathbf{w}}_{-n}$ is the estimate of the parameters without the nth training example.

The mean LOOCV error for the Olympic men's 100 m data can be seen in Figure 1.15. This plot suggests that a 3rd order polynomial would be best. This is in disagreement with the value obtained from using the last few data points as a validation set. Disagreement like this is not uncommon – **model selection** is a very difficult problem. However, the two methods do agree on one thing – the model certainly shouldn't be 6th order or above.

One drawback of illustrating model selection on a real dataset is that we don't know what the 'true' model is and therefore don't know if our selection techniques are working. We can overcome this by generating a synthetic dataset. Fifty input-target pairs were generated from a noisy third-order polynomial function and used to learn polynomial functions of increasing order (from 1st to 7th). Ideally, we hope to see minimum validation loss for the true polynomial order of 3. A further 1000 input-target pairs were generated from the true function and are used as an independent test set with which to compute an additional, independent loss. This very large dataset will give us a good approximation to the true expected loss against which we can compare the LOOCV loss.

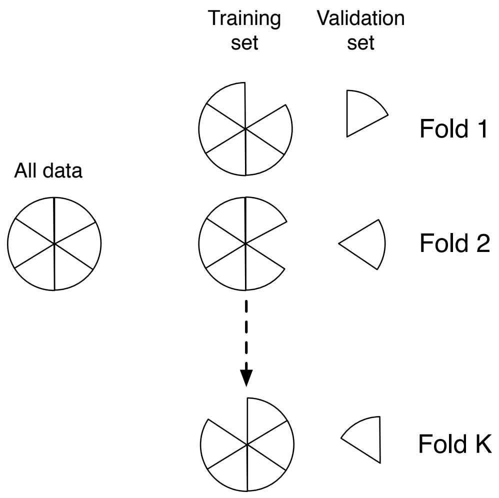


FIGURE 1.14: Cross-validation. The dataset is depicted on the left as a pie chart. In each of the K folds, one set of data points is removed from the training set and used to validate or test the model.

The results can be seen in Figure 1.16 (Matlab script: `cv_demo.m`). As we have already discovered, the training loss keeps decreasing as the order increases. The LOOCV loss and the test loss decrease as the order is increased to 3 and then increase as the order is increased further. Either of these validation methods would have predicted the correct model order. Unfortunately, we will rarely be able to call upon 1000 independent points from outside our training set and will heavily rely on a cross-validation scheme, often LOOCV.

1.5.3 Computational scaling of K-fold cross-validation

LOO cross-validation appears to be a good means of estimating our expected loss from the training data, allowing us to explore and assess various alternative models. However, consider implementing LOOCV. We need to train our model N times which will take roughly N times longer than training it once on all of the data (it is not exactly the same as we will be training it on one fewer data point). For some models, particularly if we have a lot of data, this might be infeasible.

This simplest way to alleviate this problem is to use $K \ll N$. For example, in 10-fold cross-validation, we would leave out 10% of the data for validation and use the remaining 90% for training. This reduces the number of training loops from N to 10 – a considerable saving if $N \gg 10$. A popular choice is to use N-fold cross-validation and repeat it several times with the data partitioned differently into the N groups allowing averages to be taken across both folds and repetitions.

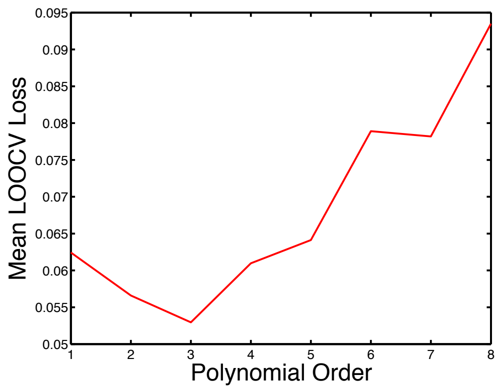


FIGURE 1.15: Mean LOOCV loss as polynomials of increasing order are fitted to the Olympic men's 100 m data.

1.6 Regularised least squares

In the previous section, we discussed how predictions on data that was not part of the training set could be used to ensure good predictive performance (good generalisation) and prevent the model from over-fitting. In essence, this stops our model becoming too complex. However, there is another way that this can be done, known as **regularisation**.

Consider a trivial model, defined by $f(\mathbf{x};\mathbf{w}) = \mathbf{w}^\mathsf{T}\mathbf{x}$ where $\mathbf{w} = [0, 0, \ldots, 0]^\mathsf{T}$ – the model always predicts a value of 0. This is the simplest model possible. Any change we make to the elements of $\mathbf{w}$ increases their absolute value and makes the model more complex. Specifically, consider the 5th order polynomial model

$$f(x;\mathbf{w}) = w_0 + w_1x + w_2x^2 + w_3x^3 + w_4x^4 + w_5x^5.$$

If we start with all of the elements of $\mathbf{w}$ being zero, the function always predicts a value of zero. Now imagine we set w_0 to some non-zero value. The model now predicts a constant (w_0). Leaving w_0 at its new value, we can set w_1 to some value. The model has become more complex and as each additional parameter is given a non-zero value, the model becomes more complex still. In general, we could consider that the higher the sum of the absolute values in $\mathbf{w}$, the more complex the model (note that it is the absolute value – we don't want the positive values to cancel with the negative ones). Alternatively, because absolute values tend to make the maths a bit harder, we could define the complexity of our model as

$$\sum_i w_i^2$$

or, in vector form,

$$\mathbf{w}^\mathsf{T}\mathbf{w}.$$

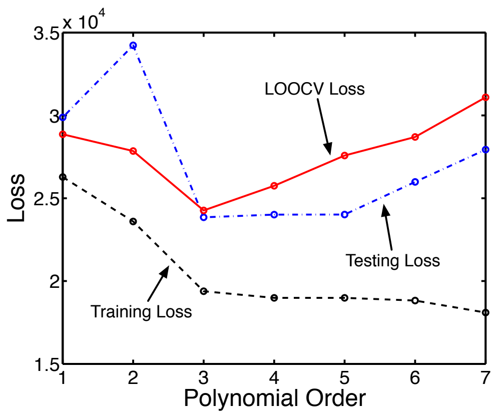


FIGURE 1.16: The Training, Testing and Leave-One-Out loss curves obtained for a noisy cubic function where a sample size of 50 is available for training and LOOCV estimation. The test error is computed using 1000 independent samples.

As we don't want our model to become too complex, it makes sense to try and keep this value low. So, rather than just minimising the average squared loss $\mathcal{L}$, we could minimise a regularised loss $\mathcal{L}'$ made by adding together our previous loss and a term penalising over-complexity:

$$\mathcal{L}' = \mathcal{L} + \lambda \mathbf{w}^\mathsf{T}\mathbf{w} \tag{1.20}$$

The parameter λ controls the trade-off between penalising not fitting the data well ($\mathcal{L}$) and penalising overly complex models ($\mathbf{w}^\mathsf{T}\mathbf{w}$). We can find the optimal value of $\mathbf{w}$ in exactly the same way as before. Adding the regularisation term to our original squared loss (Equation 1.14) gives:

$$\mathcal{L}' = \frac{1}{N}\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w} - \frac{2}{N}\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{t} + \frac{1}{N}\mathbf{t}^\mathsf{T}\mathbf{t} + \lambda\mathbf{w}^\mathsf{T}\mathbf{w}.$$

Taking partial derivatives with respect to $\mathbf{w}$

$$\frac{\partial \mathcal{L}'}{\partial \mathbf{w}} = \frac{2}{N}\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w} - \frac{2}{N}\mathbf{X}^\mathsf{T}\mathbf{t} + 2\lambda\mathbf{w}.$$

Setting this expression to zero and solving for $\mathbf{w}$ gives

$$\begin{aligned} \frac{2}{N}\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w} - \frac{2}{N}\mathbf{X}^\mathsf{T}\mathbf{t} + 2\lambda\mathbf{w} &= 0 \\ (\mathbf{X}^\mathsf{T}\mathbf{X} + N\lambda\mathbf{I})\mathbf{w} &= \mathbf{X}^\mathsf{T}\mathbf{t}. \end{aligned}$$

Hence, the reguarised least squares solution is given by:

$$\widehat{\mathbf{w}} = (\mathbf{X}^\mathsf{T}\mathbf{X} + N\lambda\mathbf{I})^{-1}\mathbf{X}^\mathsf{T}\mathbf{t}. \tag{1.21}$$

Clearly, if $\lambda = 0$ we retrieve the original solution. We can see the effect of increasing λ with a synthetic example. Figure 1.17 shows 6 synthetic data points. A fifth order polynomial function can fit the 6 data points exactly and we can see this if we set $\lambda = 0$ (in general N data-points can be perfectly fitted by a $(N-1)$th order polynomial). If we increase λ we begin to see the regularisation taking effect. $\lambda = 1e-06$ follows the general shape of the exact fifth order polynomial but without as much variability and subsequently is further from the data points. $\lambda = 0.01$ and $\lambda = 0.1$ continue this trend – the function becomes less complex (Matlab script: `regls.m`).

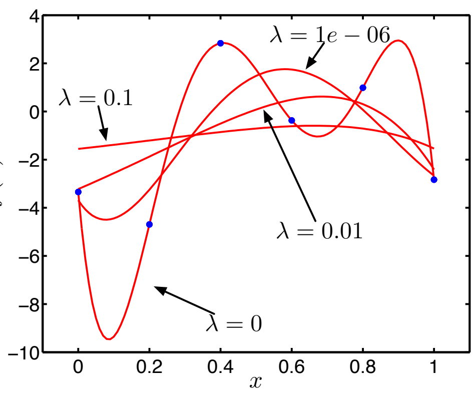


FIGURE 1.17: Effect of varying the regularisation parameter λ for a fifth order polynomial function.

Choosing the value of λ presents us with the same over-fitting/generalisation trade-off we had when choosing the polynomial order. If it is too small, our function is likely to be too complex. Too large, and we will not capture any useful trends in the data. Fortunately, we can use exactly the validation techniques introduced in the previous section to determine the best value of λ. In particular, it is common to use cross-validation to choose the value of λ that gives the best predictive performance (see Exercise EX 1.12).

1.7 Exercices

EX 1.1. By examining Figure 1.1 estimate the kind of values we should expect for w_0 and w_1 (e.g. High? Low? Positive? Negative?).

EX 1.2. Write a Matlab script that can find w_0 and w_1 for an arbitrary dataset of x_n, t_n pairs.

EX 1.3. Show that:

$$\mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w} = w_0^2\left(\sum_{n=1}^{N} x_{n1}^2\right) + 2w_0w_1\left(\sum_{n=1}^{N} x_{n1}x_{n2}\right) + w_1^2\left(\sum_{n=1}^{N} x_{n2}^2\right),$$

where

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \ \mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ \vdots & \vdots \\ x_{N1} & x_{N2} \end{bmatrix}.$$

(Hint – it's probably easiest to do the $\mathbf{X}^\mathsf{T}\mathbf{X}$ first!)

EX 1.4. Using $\mathbf{w}$ and $\mathbf{X}$ as defined in the previous exercise, show that $(\mathbf{X}\mathbf{w})^\mathsf{T} = \mathbf{w}^\mathsf{T}\mathbf{X}^\mathsf{T}$ by multiplying out both sides.

EX 1.5. When multiplying a scalar by a vector (or matrix), we multiply each element of the vector (or matrix) by that scalar. For $\mathbf{x}_n = [x_{n1},\ x_{n2}]^\mathsf{T}$, $\mathbf{t} = [t_1, \ldots, t_N]^\mathsf{T}$, $\mathbf{w} = [w_0,\ w_1]^\mathsf{T}$ and

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\mathsf{T} \\ \mathbf{x}_2^\mathsf{T} \\ \vdots \\ \mathbf{x}_N^\mathsf{T} \end{bmatrix}$$

show that

$$\sum_n \mathbf{x}_n t_n = \mathbf{X}^\mathsf{T}\mathbf{t}$$

and

$$\sum_n \mathbf{x}_n\mathbf{x}_n^\mathsf{T}\mathbf{w} = \mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{w}.$$

EX 1.6. Using the data provided in Table 1.3, find the linear model that minimises the squared loss.

EX 1.7. Using the model obtained in the previous exercise, predict the Women's winning time at the 2012 and 2016 Olympic games.

EX 1.8. Using the models for the men's and Women's 100 m, find the Olympic games when it is predicted for women to run a faster winning time than men. What are the predicted winning times? Do they seem realistic?

EX 1.9. Load the data stored in the file `synthdata.mat`. Fit a fourth order polynomial function – $f(x;\mathbf{w}) = w_0 + w_1x + w_2x^2 + w_3x^3 + w_4x^4$ – to this data. What do you notice about w_2 and w_4? Use 10-fold Cross-Validation to choose the polynomial order (between 1 and 4).

EX 1.10. Derive the optimal least squares parameter value, $\widehat{\mathbf{w}}$ for the total training loss:

$$\mathcal{L} = \sum_{n=1}^{N}(t_n - \mathbf{w}^\mathsf{T}\mathbf{x}_n)^2.$$

How does the expression compare with that derived from the average loss?

EX 1.11. The following expression is known as the *weighted* average loss:

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N} \alpha_n (t_n - \mathbf{w}^{\mathsf{T}}\mathbf{x}_n)^2$$

where the influence of each data-point is controlled by its associated α parameter. Assuming that each α_n is fixed, derive the optimal least squares parameter value $\widehat{\mathbf{w}}$.

EX 1.12. Using K-fold Cross Validation, find the value of λ that gives the best predictive performance on the Olympic men's 100 m data for (a) a first order polynomial (i.e. the standard linear model) and (b) a fourth order polynomial.

Further reading

[1] F. Galton. Regression towards mediocrity in hereditary stature. *Anthopological Miscellanea*, 15:246–263, 1886.

The term ‘regression’ was first used in the context of genetics by Francis Galton. This is one of Galton’s original genetics papers on regression from 1886.

[2] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, second edition edition, 2009.

This book includes a detailed Chapter on least squares techniques which would be a good starting point to explore this area further.

[3] K. B. Petersen and M. S. Pedersen. The matrix cookbook. http://www2.imm.dtu.dk/pubdb/p.php?3274, October 2008.

An excellent free resource that provides many useful matrix identities. Particularly useful for manipulating, and taking expectations with respect to, multi-variate Gaussian densities.