1 Multivariate linear regression

Contents

1.1 Representation: Multiple Features

Notation. n :=number of features,
$\mathbf{x}^{(i)} := x^{(i)}$ =input (feature) of i^{th} training example,
$x_j^{(i)}$:=value of feature j in i^{th} training example.

Remark. The more information we have about a data set, the better the predictions will be.

Example 1.1. Suppose now we have more features of a house on which to estimate the price of:

Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

Figure 1: Here $n = 4$, $x^{(2)} = [1416, 3, 2, 40]^T$ and so $x_2^{(2)} = 3$, for instance. So the hypothesis function extends to $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$ or $[\theta_0, \theta_1, \theta_2, \theta_3, \theta_4]\,[x_0, x_1, x_2, x_3, x_4]^T$ i.e. $x_0 = 1$, the intercept.

Remark. The general hypothesis function for n-features (excluding the additional 0^{th} feature/intercept), for one training example is

$$\begin{aligned} h_\theta(x) &= \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \dots & \theta_n \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \\ &= \boldsymbol{\theta}^T \mathbf{x} \end{aligned}$$

For all m training examples, we use the notation

$$\begin{aligned} X &:= \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \cdots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_0^{(m)} & x_1^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \\ h_\theta(X) &= X\boldsymbol{\theta} \end{aligned}$$

where $x_0^{(i)} = 1, \forall i = 1, ..., m$.

Definition 1.1. The **design matrix** of a training example is given by the matrix $X \in \mathbb{R}^{m,n+1}$ above.

1.2 Evaluation: Cost function for multivariate linear regression

$J : \mathbb{R}^{n+1} \longrightarrow \mathbb{R}$ where

$$\begin{aligned} J(\theta) &= \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2 \\ &= \frac{1}{2m} \left[X\boldsymbol{\theta} - \mathbf{y} \right]^T \left[X\boldsymbol{\theta} - \mathbf{y} \right] \end{aligned} \tag{1}$$

for $\mathbf{y} \in \mathbb{R}^m$, $\boldsymbol{\theta} \in \mathbb{R}^{n+1}$, $X \in \mathbb{R}^{n \times n}$.

1.3 Optimisation: Gradient Descent for multivariate linear regression

Algorithm 1. ***Gradient Descent***
repeat until convergence {
$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \nabla J(\boldsymbol{\theta})$
}
for α some (positive) ***learning rate*** *(aka. step size) and some initial guess for $\boldsymbol{\theta}$.*

Remark. $\nabla J(\boldsymbol{\theta}) = \left[\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_0}, \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1}, ..., \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_n}\right]^T$ where $\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}\left(h_{\boldsymbol{\theta}}\left(x^{(i)}\right) - y^{(i)}\right)$. Equally $\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \frac{1}{m}\mathbf{x}_j^T(X\boldsymbol{\theta} - \mathbf{y})$, so $\nabla J(\boldsymbol{\theta}) = \frac{1}{m}X^T(X\boldsymbol{\theta} - \mathbf{y})$ and $\boldsymbol{\theta} := \boldsymbol{\theta} - \frac{\alpha}{m}X^T(X\boldsymbol{\theta} - \mathbf{y})$; representing sums as a matrix is very useful, as there exist highly optimised libraries in performing matrix multiplication on a computer (e.g. the multiplication steps/updates can be computed in parallel).

1.4 Gradient Descent in practice

1.4.1 Feature Scaling

Problem

We should make sure all features have a *similar scale* so that gradient descent will converge more quickly. Consider the example in figure 1; plotting the contours for $w_1 := \theta_1$ vs. $w_2 := \theta_2$ gives a very tall and thin shape due to the large range difference between the two parameters:

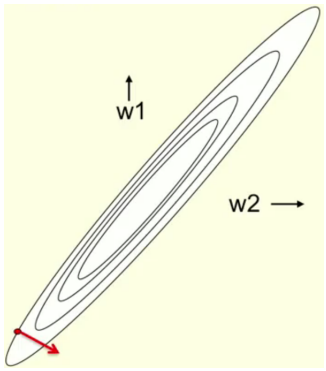

Figure 2: An elongated ellipse has direction of steepest descent almost perpendicular to the direction towards the minimum; the red gradient vector has a large component along the short axis of the ellipse (very steep descent), and a small component along the long axis of the ellipse - just the opposite to what we want.

Solution

Feature Scaling:

As a rule of thumb, acceptable ranges for our input variables (features) are:

- $-3 \leq x_j^{(i)} \leq 3$ (no bigger)
- $-\frac{1}{3} \leq x_j^{(i)} \leq \frac{1}{3}$ (no smaller)

To achieve this, we can use **mean normalisation** *for each* training example

$$\hat{x}_j = \frac{x_j - \mu}{s}$$

where μ :=average of all features $(x_1, ..., x_n)$ and $s := \max_{j=1,..,n} x_j - \min_{j=1,..,n} x_j$.

Note. **WARNING:** We do not scale $x_0 = 1$.

Remark. s can also be replaced with the standard deviation.

1.4.2 Debugging gradient descent/Judging convergence

If gradient descent is working then $J(\boldsymbol{\theta})$ should *decrease after every iteration.*

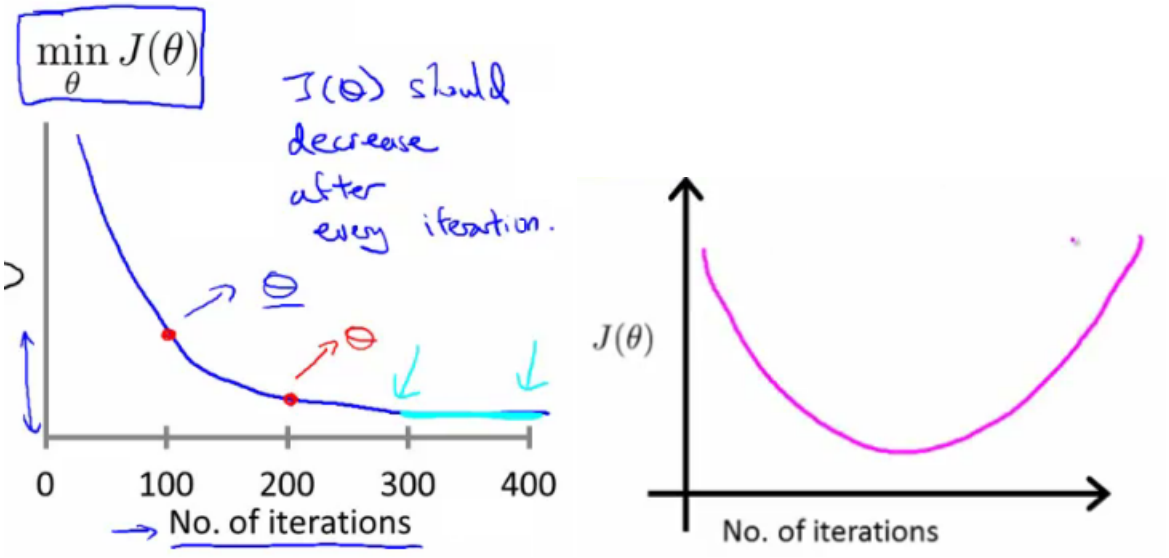

Figure 3: It appears that GD is working effectively on the left; convergence has been reached by approximately 300 iterations. If $J(\boldsymbol{\theta})$ ever increases (e.g. right picture), it is likely to mean that we have overshot the minimum and so a smaller α is required.

Remark. In general, it is hard to establish in advance how many iterations will be needed (anywhere from $30 - 3,000,000$ iterations). Typically declare convergence if $J(\boldsymbol{\theta})$ decreases by less than 10^{-3} in one iteration (or via an observation like that in figure 3).

1.4.3 Choosing a suitable lesimplified arning rate α

Typically run gradient descent on a range of α values, separating them by roughly threefold difference (e.g. 0.001,0.003,0.01,0.03,0.1,...). Problems that can occur by choosing the wrong α:

- *Too big:* GD can overshoot the minimum (figure 3).
- *Too small:* GD can be slow to converge.

1.4.4 Features and Polynomial Regression

Our hypothesis function need not be linear if that does not fit the data well. We can transform the behaviour (or curve) of our hypothesis function by making it a quadratic, cubic, square root function or any other form; e.g. $h_\theta(x) = \theta_0 + \theta_1 x_1$ could become:

- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$
- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$
- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$
- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \log(x_1)$

Note. The parameters $(\theta_0, \theta_1, ...)$ still exist in a linear manner.

To map our old linear hypothesis and cost functions to these polynomial descriptions, the easy thing to do is set e.g. $x_1 := x_1$, $x_2 := \sqrt{x_1}$ so that $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$; note to scale, we scale only w.r.t. x_1, so $\sqrt{\frac{x_1}{\max_j x_j}}$.

Remark. The ability to enter features in non-linear forms makes linear regression a very powerful model that has much widespread use; indeed, non-linear relationships can often be closely approximated by Taylor series expansions.

1.5 Alternative to gradient descent: normal equations

Iterative algorithms (such as gradient descent) takes steps to converge; normal equation solves for the optimum θ analytically.

Gradient Descent	Normal Equations
Need to choose α	No need to choose α
Choose the number of iterations (or can require many)	No iterations involved
Works well when n is large	Slow if n is very large (matrix inversion is typically $\mathcal{O}\left(n^3\right)$)

Table 1: Comparison of gradient descent and normal equations.

Remark. Roughly when n exceeds $10,000$ is a good point for changing between the two methods ("Just a thought: could we combine the two methods/use a hybrid version at this point of intersection???")

1.5.1 Cost function for normal equations

Taking the cost function as defined in equation 1, using the usual method (calculus) to find the minimum (see `https://en.wikipedia.org/wiki/Linear_least_squares_%28mathematics%29#Derivation_of_the_normal_equations` for a proof - note $S \propto J(\boldsymbol{\theta})$), we see that

$$\boldsymbol{\theta} = \left(X^T X\right)^{-1} X^T \mathbf{y}$$

Note. For any matrix $Y \in \mathbb{R}^{m,n}$, $Y^T Y$ is square and so invertible (assuming its determinant is non-zero).

Remark. If $X^T X$ is not invertible, it is likely to be due to:

- ***Redundant features (aka. Collinearity)*** Where two features are very closely related (i.e. they are linearly dependent).
- ***Too many features (e.g. $m \leq n$)*** In this case, delete some features or use "regularisation" (let's you use lots of features for a small training set - see later notes).