Error Functions & Linear Regression (1)

John Kelleher & Brian Mac Namee

Machine Learning @ DIT

Univariate Linear Regression

- There are a family of machine learning approaches that can be thought of as directly searching for a set of parameters that maximise the performance of a particular prediction model.
- In this lecture we will look at linear regression and how it can be used for both categorical and continuous prediction problems.

Linear Regression

- Remember that the equation of a line, $y = mx + b$, can be used to represent the relationship between two variables, x and y.
- For simplicity we will rewrite the equation of a line as:

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

where $\mathbf{w}$ is the vector $[w_0, w_1]$ and w_0 and w_1 are referred to as weights.

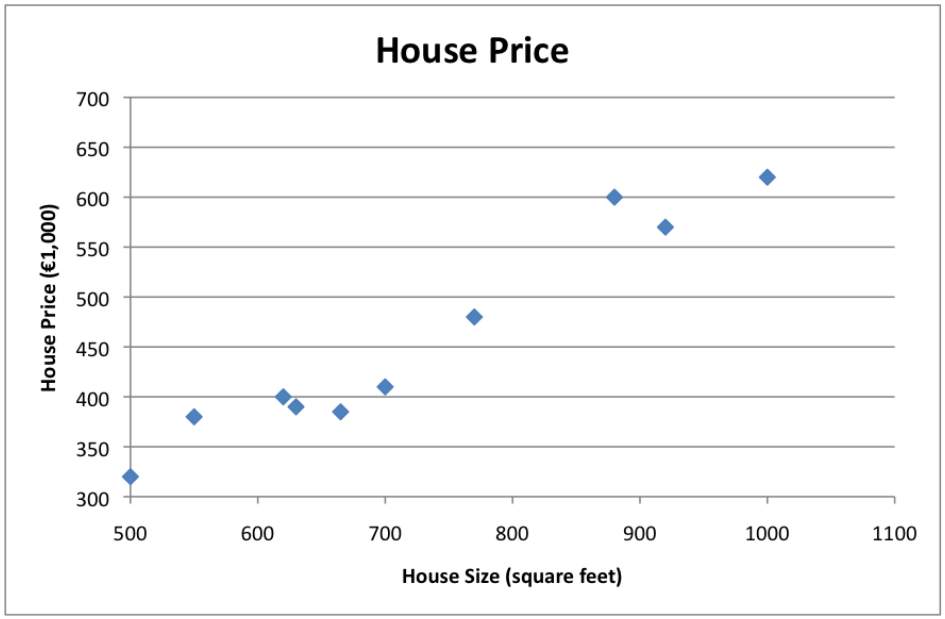
- We now just need to find the weight values that maximise the performance of this model - i.e. the weight values that allow us best predict y given x.

Loss Function

- In order to find the best values for $[w_0, w_1]$ we can measure the fit of a model by measuring squared loss function, L_2 over all training examples
- The loss function can be defined as:

$$\begin{aligned} Loss(h_{\mathbf{w}}) &= \sum_{j=1}^{N} L_2(y_j, h_{\mathbf{w}}(x)) \\ &= \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(x))^2 \\ &= \sum_{j=1}^{N} (y_j - (w_0 + w_1 x_j))^2 \end{aligned}$$

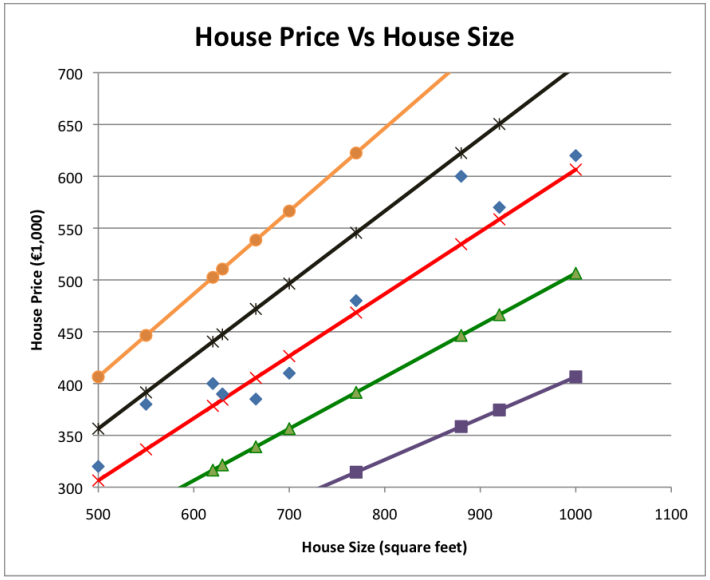
House Size	House Price
500	320
550	380
620	400
630	390
665	385
700	410
770	480
880	600
920	570
1000	620

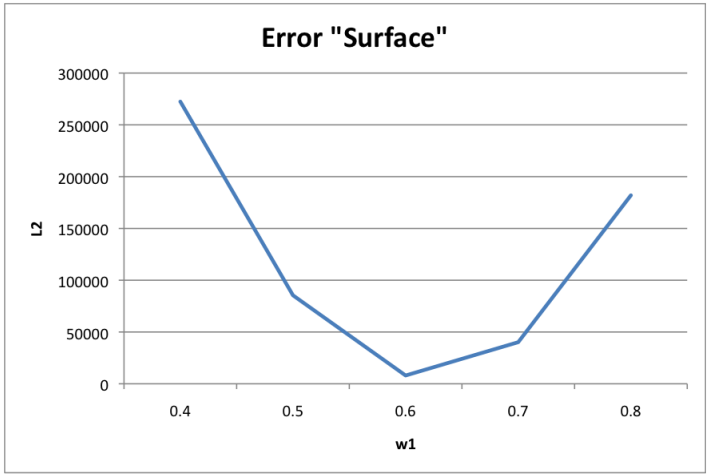


Practice

Assuming $w_0 = 6.47$ what would be the most accurate value of w_1 based on the given training set (select from 0.4, 0.5, 0.6, 0.7, 0.8).

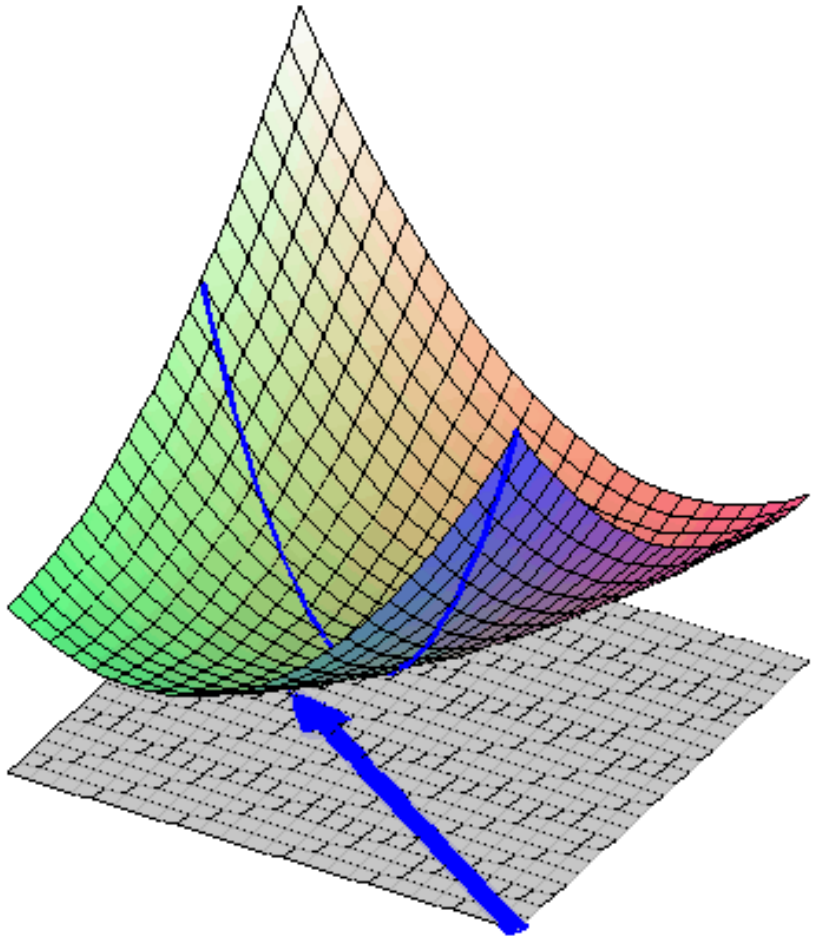
w_1	L_2
0.4	272,436
0.5	85,424
0.6	8,006
0.7	40,183
0.8	181,955





- The *Loss* depending on the weights $[w_0, w_1]$ can be viewed as a surface - in fact a surface that is convex and that has a global minimum.
- We need to find the lowest point on this surface and use it as our weight values.

Linear Regression



- So we want to find $\mathbf{w}^* = argmin_{\mathbf{w}} Loss(h_{\mathbf{w}})$.
- The sum $Loss(h_{\mathbf{w}})$ is minimised when its partial derivatives with respect to w_0 and w_1 are equal to 0.

- So we want to find $\mathbf{w}^* = argmin_{\mathbf{w}} Loss(h_{\mathbf{w}})$.
- The sum $Loss(h_{\mathbf{w}})$ is minimised when its partial derivatives with respect to w_0 and w_1 are equal to 0.

Partial What????

Who knows what a derivative is?

Who knows what a partial derivative is?

In order to compute slopes we need to know some calculus:

Fundamental Calculus:

- In mathematics, the derivative is a measurement of how a function changes when the values of its inputs change and a partial derivative (denoted by the symbol ∂) of a function of several variables is its derivative with respect to one of those variables with the others held constant.

- There are really only three things we need to know about calculus for working on linear regression. First:

$$\frac{\partial}{\partial x} x = 1$$

$$\frac{\partial}{\partial x} x^2 = 2x$$

Practice

- Calculate the following:

$$\frac{\partial}{\partial x} x + 1 =$$

$$\frac{\partial}{\partial x} 6x =$$

$$\frac{\partial}{\partial x} x^2 =$$

$$\frac{\partial}{\partial x} 3x^2 =$$

$$\frac{\partial}{\partial x} x^2 + 2x + 4 =$$

Practice

- Calculate the following:

$$\frac{\partial}{\partial x} x + 1 = 1$$

$$\frac{\partial}{\partial x} 6x = 6$$

$$\frac{\partial}{\partial x} x^2 = 2x$$

$$\frac{\partial}{\partial x} 3x^2 = 6x$$

$$\frac{\partial}{\partial x} x^2 + 2x + 4 = 2x + 2$$

Fundamental Calculus:

- The second thing is the Chain Rule
- The Chain Rule allows us calculate the derivative of a composition of two or more functions.
- Chain rule: $\frac{\partial}{\partial x} g(f(x)) = \frac{\partial}{\partial f(x)} g(f(x)) \frac{\partial}{\partial x} f(x)$
- Ex.1: $\frac{\partial}{\partial x}(x^2 + 1)^2 = 2(x^2 + 1)(2x)$
- Ex.2: $\frac{\partial}{\partial x} \frac{1}{2}(c - x)^2 =$

  $\frac{2}{2}(c - x)(\frac{\partial}{\partial x}(c - x)) = (c - x)(-1) = -(c - x)$

Practice

- $\frac{\partial}{\partial x}(x^2 + 2)^2 =$
- $\frac{\partial}{\partial x}(3x^2 + 3x + 3)^2 =$

Practice

- $\frac{\partial}{\partial x}(x^2+2)^2 = 2(x^2+2)(2x)$
- $\frac{\partial}{\partial x}(3x^2+3x+3)^2 = 2(3x^2+3x+3)(6x+3)$

- So we want to find $\mathbf{w}^* = argmin_{\mathbf{w}} Loss(h_{\mathbf{w}})$.
- The sum $Loss(h_{\mathbf{w}})$ is minimised when its partial derivatives with respect to w_0 and w_1 are equal to 0.
- So:

$$\frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - (w_0 + w_1 x_j))^2 = 0$$

and

$$\frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_0 + w_1 x_j))^2 = 0$$

- Fortunately, these equations have a unique solution*:

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}$$

and

$$w_0 = \frac{(\sum y_j - w_1(\sum x_j))}{N}$$

*based on assumptions that there is normally distributed noise and the stationarity assumption amongst other things

Practice

House Size	House Price
500	320
550	380
620	400
630	390
665	385
700	410
770	480
880	600
920	570
1000	620

- Let's work out the optimal values for $[w_0, w_1]$ for the house price dataset we looked at previously.
- See the associated Excel sheet for the solution!

- While it was great to have a unique solution to our system of equations in the previous example, this will not always be the case.
- We need an alternative way to find the best weights to use in our model.
- **Any ideas?**

- We need to perform a search through the weight space for the best values to use.
- An exhaustive search is just not reasonable.
- Instead we can use the common approach of gradient descent:
  - Start at any point in weight space
  - Move to a neighbouring point that is *downhill*
  - Repeat until we converge on the point of minimum loss.

Gradient descent

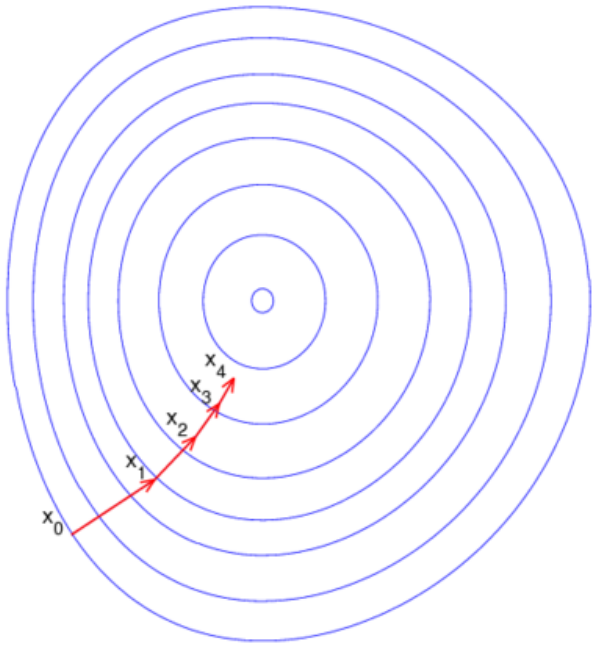
$\mathbf{w} \leftarrow$ any point in the parameter space

loop until convergence **do**

for each w_i **in w do**

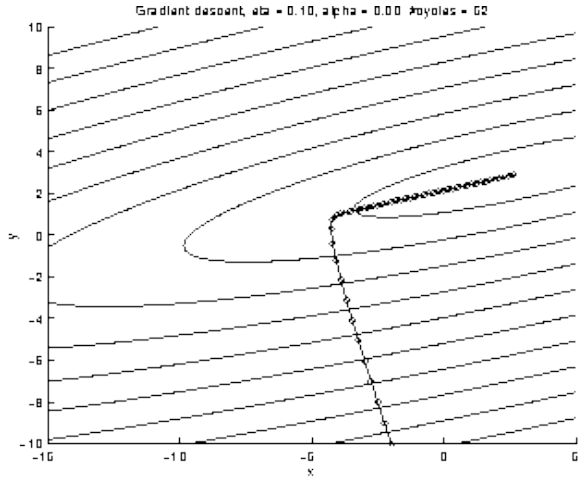
$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

- α is referred to as the learning rate and determines the amount by which weights are changed at each step.
- This can be a fixed constant or can decay over time as learning proceeds.

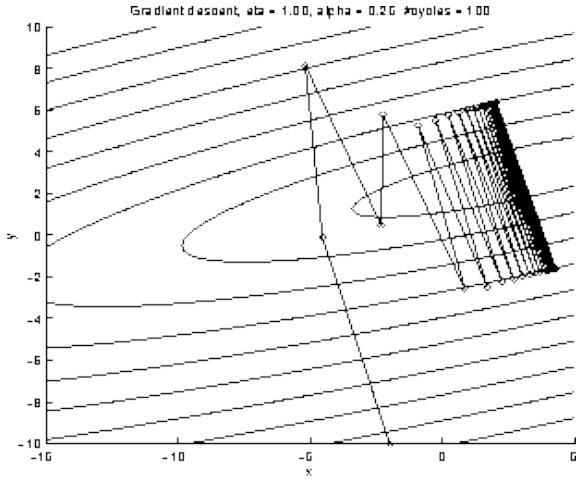


Nice gradient descent animations: http://animation.yihui.name/compstat:gradient_descent_algorithm

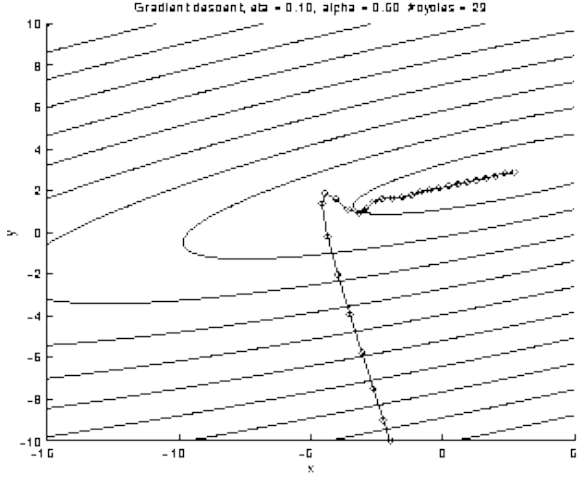
Gradient Descent



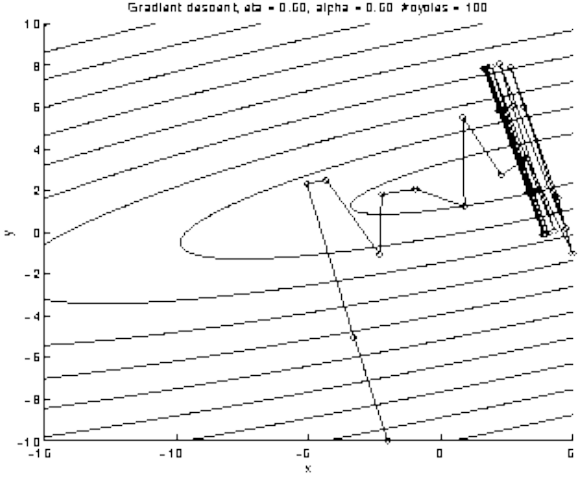
(a)



(b)



(c)



(d)

Imagine there was just one training example (x, y). Then the gradient would be given as (note use of chain rule):

$$\begin{aligned}\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))^2 \\ &= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x)) \\ &= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_0 + w_1 x))\end{aligned}$$

Applying to both w_0 and w_1 we get:

$$\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x))$$
$$\frac{\partial}{\partial w_1} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x$$

Plugging these into our weight update rule we get:

$$w_0 \leftarrow w_0 + \alpha(y - h_{\mathbf{w}}(x))$$
$$w_1 \leftarrow w_1 + \alpha(y - h_{\mathbf{w}}(x)) \times x$$

Intuitively this makes sense:

- if $h_{\mathbf{w}}(x)$ is too high reduce w_0 a little bit and reduce w_1 if x was positive but increase it if x was negative.
- if $h_{\mathbf{w}}(x)$ is too low do the opposite.

The previous example only considered a single training example.
However, adjusting this to deal with a full training set of N examples simply involves introducing a sum:

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j))$$

$$w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \times x_j$$

This is known as batch gradient descent.

An alternative to batch gradient descent is stochastic gradient descent in which the update rule is applied one example at a time and the examples are chosen, usually randomly, from the training set.

Multivariate Linear Regression

- We have seen how linear regression can be used to make continuous predictions from a single variable.
- However, we like big data so one variable is no good to us!
- A multivariate linear regression does the same job using multiple variables.

Multi-variate Linear Regression

- Our hypothesis space is of the form:

$$\begin{aligned} h_{\mathbf{w}}(x) &= w_0 + w_1 x_1 + \cdots + w_n x_n \\ &= w_0 + \sum_i w_i x_i \end{aligned}$$

- We can make this look a little neater by inventing a dummy input attribute, x_0, that is always equal to 1. Then:

$$\begin{aligned} h_{\mathbf{w}}(x) &= \sum_i w_i x_i \\ &= \mathbf{w} \cdot \mathbf{x} \end{aligned}$$

Gradient Descent for Multi-variate Linear Regression

- The best set of weights, $\mathbf{w}^*$ minimizes the squared error loss, L_2, over a set of training examples:

$$\mathbf{w}^* = argmin_{\mathbf{w}} \sum_j L_2(y_j, \mathbf{w} \cdot \mathbf{x}_j)$$

- Gradient descent works for the multi-variate case just like it does for the univariate case and will minimise the loss function.
- The update rule for each weight is:

$$w_i \leftarrow w_i + \alpha \sum_j x_{j,i}(y_i - h_{\mathbf{w}}(\mathbf{x}_j))$$

Summary