Week 3b:

Learning and extending linear regression

G6061: Fundamentals of Machine Learning [23/24]

Dr. Johanna Senk

UNIVERSITY OF SUSSEX

Recap of previous lecture

- Regression models predict real numbers from the instance features.
- Linear regression finds the solution that minimises the sum of squared difference between predictions and the labelled data.
  - This arises from describing our predictions as being corrupted by Gaussian distributed noise $\mathbf{y} = \hat{\mathbf{y}} + \epsilon$
  - As each of our pairs of instances/labels are independent, the joint probability is the product of all the individual probabilities $p(\mathbf{y}|\mathbf{x}) = \prod_i p(y_i|x_i)$.
  - This is nicer to express as a sum of log probabilities, which for a normal distribution is proportional to the sum of squared differences.
- Linear regression is called that because the model parameters ($\mathbf{w}$) are linearly combined (added) with each other to produce the output.
- One way of fitting a linear regression model is to compute the pseude-inverse.

Learning in least squares linear regression: 1st approach

The linear regression algorithm

1. Construct the matrix X and the vector $\mathbf{y}$ from the data set $(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)$ as follows

$$\underbrace{X = \begin{bmatrix} \text{—}\mathbf{x}^{1,\top}\text{—} \\ \text{—}\mathbf{x}^{2,\top}\text{—} \\ \vdots \\ \text{—}\mathbf{x}^{N,\top}\text{—} \end{bmatrix}}_{\text{input data matrix}} \quad \underbrace{\mathbf{y} = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^N \end{bmatrix}}_{\text{target vector}}$$

2. Compute the **pseudo-inverse** $X^\dagger = (X^\top X)^{-1} X^\top$
3. Return $\mathbf{w} = X^\dagger \mathbf{y}$

Outline

- Today we're going to talk about a second method for fitting linear regression models to data.
- We'll think about how to make linear regression work for a wide range of data.
- We'll also discuss a couple of applications in some more detail.

Learning in least squares linear regression: 2nd approach

Minimising residual error E_{res}

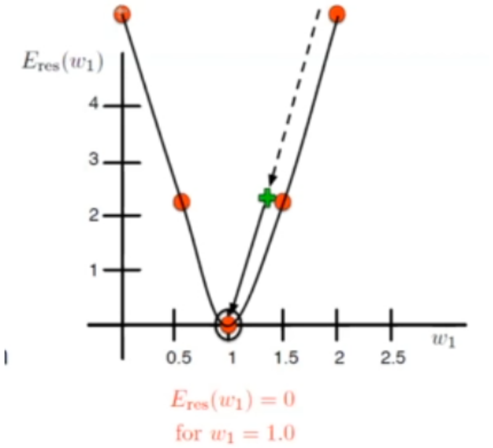
- Use **gradient descent algorithm**
- It is a general algorithm used in many applications for minimising functions
  - Have some function $J(w_0, w_1, w_2, \ldots, w_d)$
  - Want $\underset{w_0,w_1,w_2,\ldots,w_d}{\text{minimise}} J(w_0, w_1, w_2, \ldots, w_d)$

Algorithm:

- Choose initial parameter values (e.g., random or zeros)
- Repeat until convergence:
  - **Change parameters a bit to reduce a cost**
- Output minimised parameter values

Gradient descent for linear regression - 1D

- **Metaphor**: “Walking down the hill”
  - Keep taking steps down hill
  - Until cannot go downhill anymore
- This strategy makes it possible to find minimum of “bowl” shaped (**convex**) cost function



Changing the parameters $\mathbf{w}$

- Still need to know *how* to change the parameters
  - i.e. how to "walk downhill"

$$\boxed{\mathbf{w} := \mathbf{w} - r \times \underbrace{\frac{1}{N} X^\top (X\mathbf{w} - \mathbf{y})}_{\nabla_{\mathbf{w}} E_{\text{res}}(\mathbf{w})}}$$

  - N is the number of training examples (i.e. $i = 1, \ldots, N$)
  - r is the learning rate: determines the size of steps
  - rest tells us size and direction (+,−) of slope of cost function
- **NB**: model weights $\mathbf{w} = (w_0, w_1, \ldots, w_d)$ are in $\mathbb{R}^{d+1}$

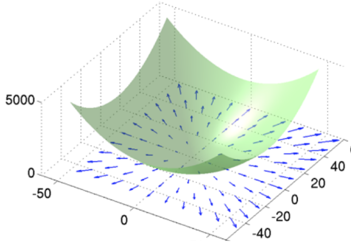


Figure is from Chris Kiefer

Implementation: batch gradient descent

1. Construct the matrix X and the vector $\mathbf{y}$ from the data set $(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)$ as follows

$$\underbrace{X = \begin{bmatrix} \text{—}\mathbf{x}^{1,\top}\text{—} \\ \text{—}\mathbf{x}^{2,\top}\text{—} \\ \vdots \\ \text{—}\mathbf{x}^{N,\top}\text{—} \end{bmatrix}}_{\text{input data matrix}} \quad \underbrace{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}$$

2. Initialise $\mathbf{w}$: $w_0 = 0, w_1 = 0, \ldots, w_d = 0$
3. Repeat until $\mathbf{w} = (w_0, w_1, \ldots, w_d)$ do not change

$$\mathbf{w} := \mathbf{w} - r \times \frac{1}{N} X^\top (X\mathbf{w} - \mathbf{y})$$

4. Return parameter values $\mathbf{w}$

Pseudo-inverse vs. gradient descent

N training instance, d features

Pseudo-inverse

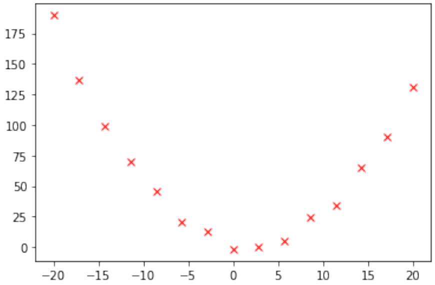
- No need to choose r
- Do not need to iterate
- Need to compute $(X^\top X)^{-1} \rightarrow$ computational complexity is $O((d+1)^3)$
- Slow if d is very large!

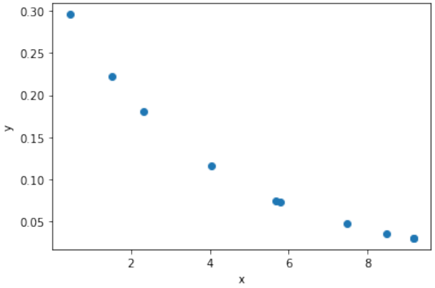
Gradient descent

- Need to choose r
- Needs many iterations
- Works well even when d is extremely large

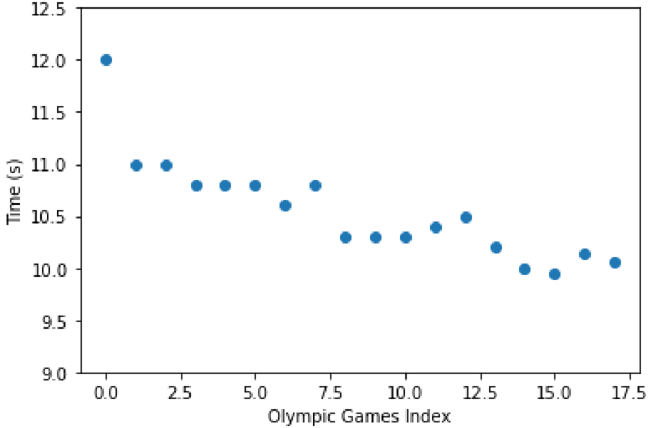
Extending linear regression

- Linear regression assumes that the dependent variable can be calculated as a weighted sum of input features.
- Linear regression implements: $y \approx \hat{f}(x_i) = \sum_{i=0}^{d} w_i x_i$
- Models are easy to learn because of the **linearity in the weighted input features**.
- Let's take a minute to think about how we could extend linear regression to describe more complex data:



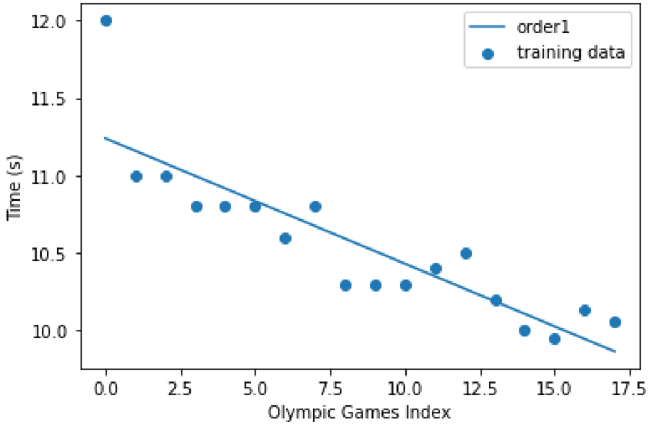


Olympic data



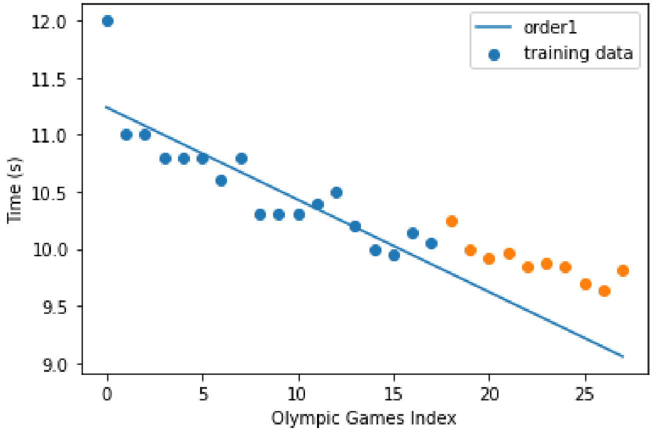
Winning men’s 100m sprint times at the Summer Olympics.
What models can we use to predict future events?

Linear regression



A straight line works quite well...

Linear regression



but maybe it's not flexible enough?

Extending linear regression

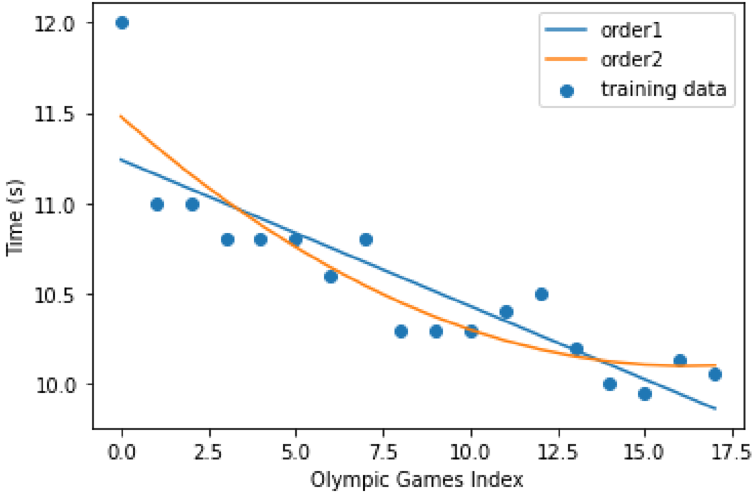
- Construct matrix X and the vector $\mathbf{y}$ from the olympic men's 100 m data $(1896, 12.00s), (1900, 11.00s), \ldots, (2008, 9.69s)$:

$$X = \begin{bmatrix} 1 & x_1 (= 1896) \\ 1 & x_2 (= 1900) \\ & \vdots \\ 1 & x_N (= 2008) \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y^1 (= 12.00) \\ y^2 (= 11.00) \\ \vdots \\ y^N (= 9.69) \end{bmatrix}$$

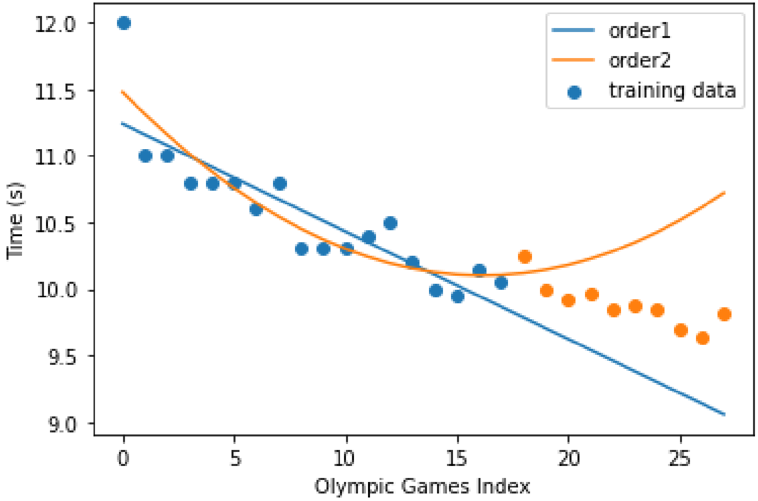
- We can augment our data matrix, X, to allow non-linear functions of the data!
- For a Kth order polynomial:

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^K \\ 1 & x_2 & x_2^2 & \ldots & x_2^K \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & \ldots & x_N^K \end{bmatrix}$$

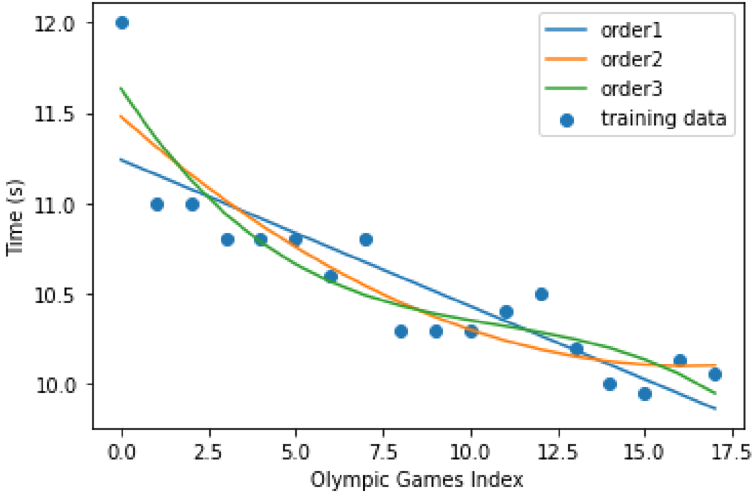
Polynomial regression



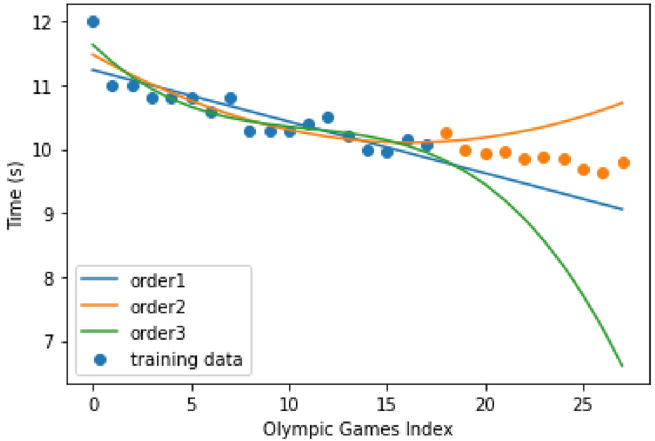
Polynomial regression



Polynomial regression

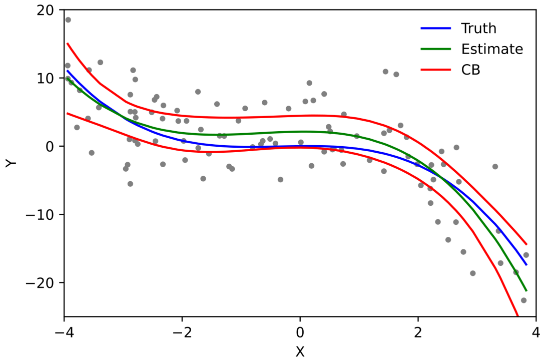


Polynomial regression



Maybe these features aren't that useful for this problem!

Polynomial regression



(Wikipedia CC BY 3.0)

But when the data is truly a polynomial they help (here: cubic)!

More generally

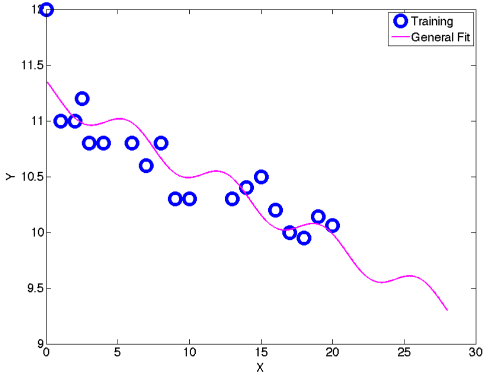
- We are not restricted to polynomial functions
- We can define any set of K functions of x, $h_k(x)$

$$X = \begin{bmatrix} h_0(x_1) & h_1(x_1) & h_2(x_1) & \dots & h_K(x_1) \\ h_0(x_2) & h_1(x_2) & h_2(x_2) & \dots & h_K(x_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ h_0(x_N) & h_1(x_N) & h_2(x_N) & \dots & h_K(x_N) \end{bmatrix}$$

- As an example:
  - $h_0(x) = 1$
  - $h_1(x) = x$
  - $h_2(x) = \sin(\frac{x-2660}{4.3})$

Thus, $\hat{f}(x, \mathbf{w}) = w_0 + w_1 x + w_2 \sin(\frac{x-2660}{4.3})$

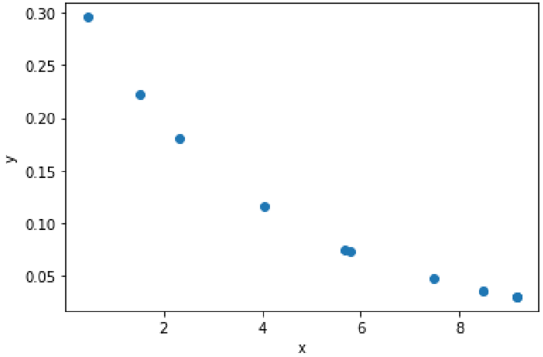
General basis functions $h(\cdot)$ fitted to the Olympics data



$$\hat{f}(x, \mathbf{w}) = w_0 + w_1 x + w_2 \sin(\tfrac{x-2660}{4.3})$$

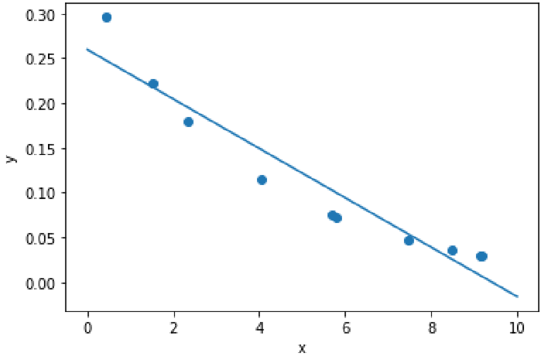
What about the label space?

- Linear least squares regression makes some restrictive approximations in how it predicts y from x.
- Imagine we have some data like:



What will linear regression do?

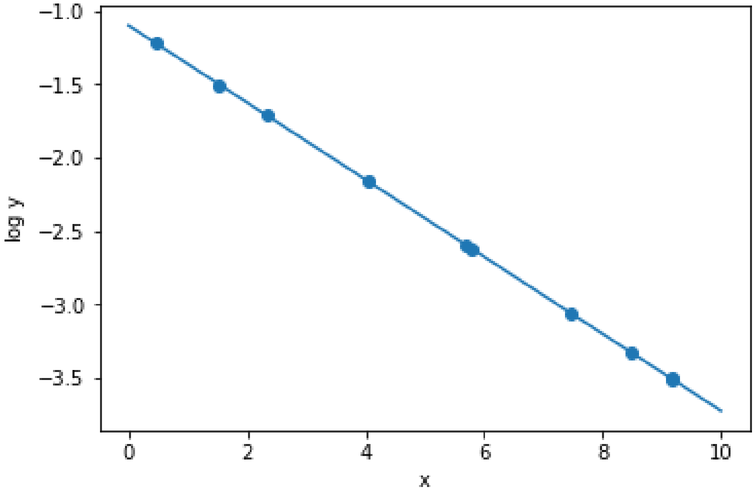
What about the label space?



Not fit very well! What could we do to fix this? What do we notice about the data?
y varies *exponentially* in x! What's a simple fix?

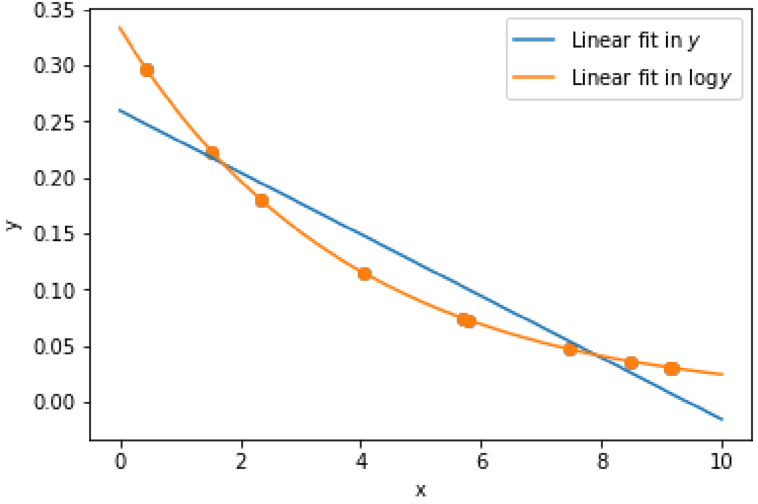
What about the label space?

If we take the log of y we get a perfect fit.



What about the label space?

Our predictive linear model now predict $\log(y)$, and fits perfectly.



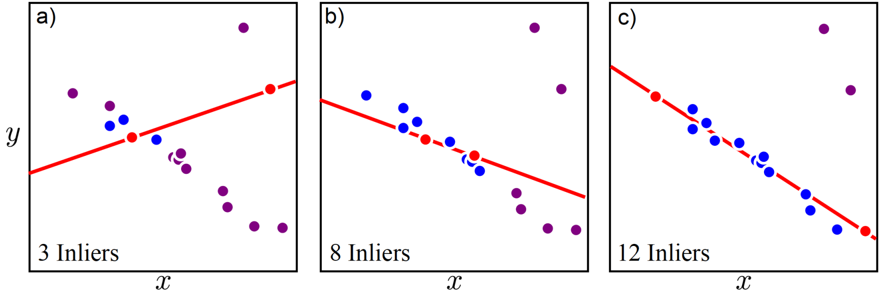
Outliers in linear regression

- Because we have assumed that our data can be explained by the model with some added Gaussian noise at some scale, we are not resistant to *outliers*.
- Outliers are often erroneous data samples that cannot be well explained by the model.
- There's a few ways to reduce the influence of outliers:
    - One is by changing the error probability distribution to something different, like a student-t distribution.
    - Another is saying our error is a **mixture of Gaussians** where most of the data has a small σ and some data has a big σ.
    - These make the maths trickier, so something like RANSAC might be easier to deal with.

RANSAC for robustness to outliers

- **RAN**dom **SA**mple **C**onsensus - RANSAC.
- The basic idea
  1. Choose a minimal random subset of your data (in our case, pairs of matching feature points).
  2. Use this subset of the data to estimate the parameters for your function (in our case, estimating the transformation between images)
  3. Compute which of the total data points agree well with the parameters we estimated, i.e. our inliers under this hypothesis (in our case, identify which of all the pairs of matching feature points support this hypothesis).
  4. Repeat steps 1-3 for a fixed time.
  5. Re-estimate the model using the largest set of inliers we found from step 3.

RANSAC for line fitting



(from S. Prince, Computer Vision: Models, Learning, and Inference)

- We only need 2 sample points to define a 2D line.
- The left and middle images show 2 possible hypothesises defined by a minimal subset of points.
- The image on the right shows the hypothesis with the maximum number of inliers.

Applications: modelling complex data

- Problems like forecasting the effects of viral epidemics or climate change are complex.
  - This is because of a variety of different and incomplete information sources, with varying levels of noise in the observations.
- Often we want to be able to understand how we arrived at the answer, as well as make predictions
  - i.e., make the model inspectable, by looking at what weights are assigned to different factors.
- There are often ambiguities in real data modelling problems, if 2 of our input data variables are highly correlated (or anti-correlated) then the model may still choose different values for each weight.

Summary and outlook

- Linear regression is a fundamental and well used tool for predicting continuous values.
- We can increase the complexity of linear regression, by treating non-linear transformations of the input data space as additional data elements.
- There are several ways to fit a linear regression model, and we have described two of them.
- We've talked about outliers in linear regression.

Next lecture:

- Model selection criteria and regularisation.