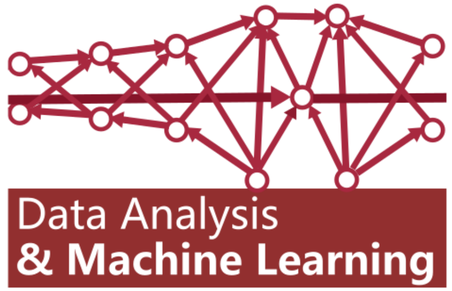


Data Analysis and Machine Learning 4 (DAML)

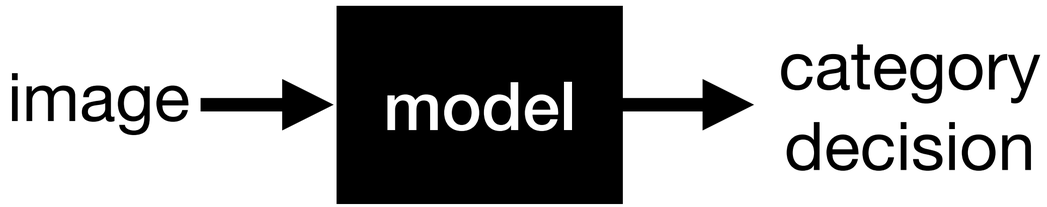
Week 5: Linear models for regression

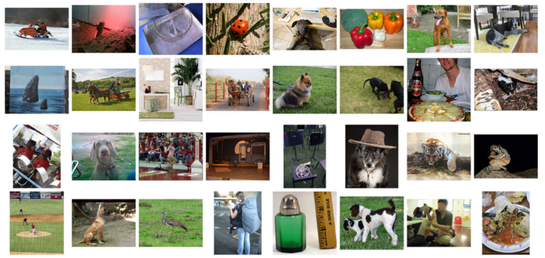
Elliot J. Crowley, 12th February 2024

THE UNIVERSITY *of* EDINBURGH

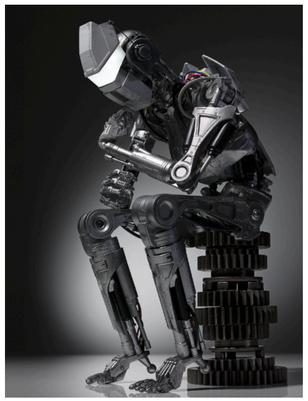
Recap

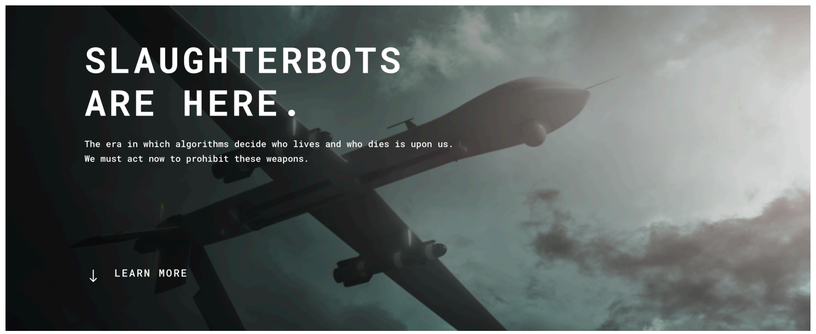
- We learned about supervised learning and looked at some examples

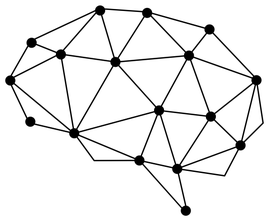




- We considered ethical issues that can arise when applying ML in society



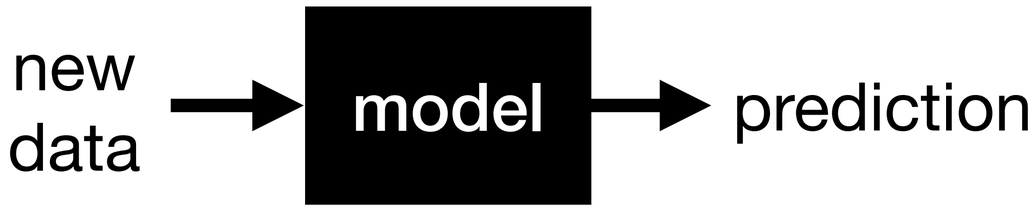




Cambridge
Analytica

Supervised Learning

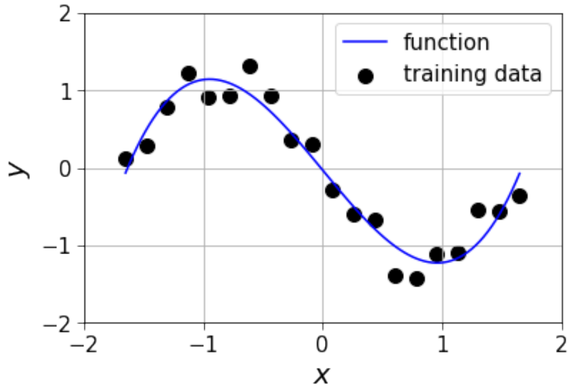
- We want a model that takes in a new data point and outputs a prediction



- For the model to be accurate it must first learn from training data
- Often, models are parameterised functions and learning = finding the best parameters
- Training data is a set of existing data points that have been **labelled**
- The label says what the prediction for that data point **should be**

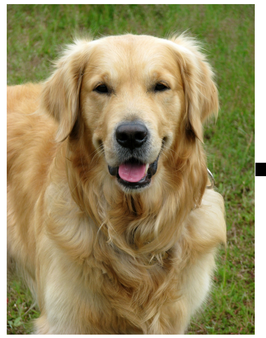
Two canonical problems in supervised learning

- **Regression**: Given input data, predict a continuous output



- **Classification**: Given input data, predict a distinct category

cat



dog

Linear models for regression

The regression problem

- Our training set consists of N data point-target pairs $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$
- Data points $\mathbf{x} \in \mathbb{R}^D$ are column vectors, targets (/labels) $y \in \mathbb{R}$ are scalar
- We can use matrix/vector notation as in Week 3

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)\top} \\ \mathbf{x}^{(2)\top} \\ \mathbf{x}^{(3)\top} \\ \vdots \\ \mathbf{x}^{(N)\top} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_D^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_D^{(2)} \\ x_1^{(3)} & x_2^{(3)} & \dots & x_D^{(3)} \\ \dots & \dots & \ddots & \vdots \\ x_1^{(N)} & x_2^{(N)} & \dots & x_D^{(N)} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \vdots \\ y^{(N)} \end{bmatrix}$$

- **Objective:** We want some function f such that $f(\mathbf{x}^{(n)}) = y^{(n)}$ for each training point. This function is our regression model

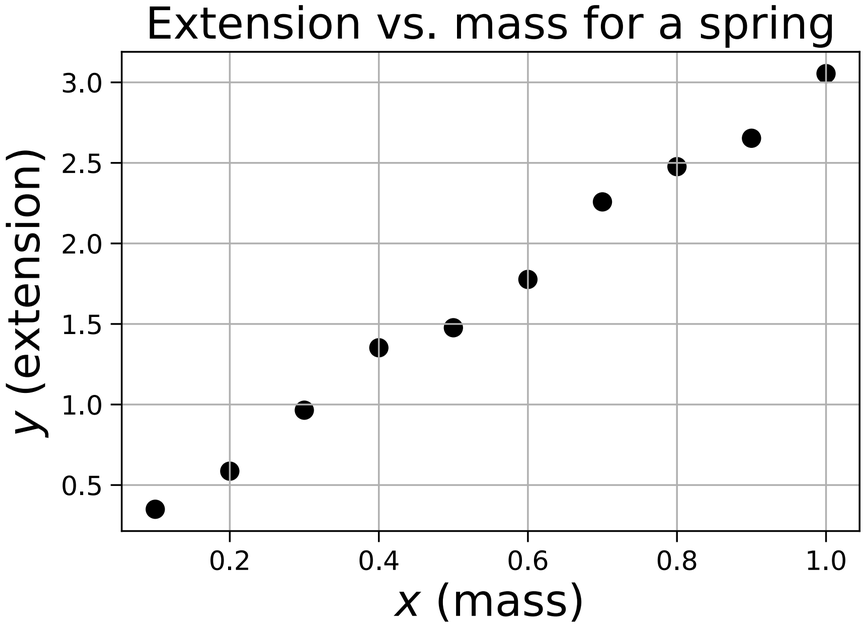
Simple linear regression

- We have 1D measurements of mass-extension pairs $\{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$
- We want a model represented by f s.t. $f(x^{(n)}) = y^{(n)}$ for each point
- Let's fit a line and denote its outputs as $\hat{y}$

$$f(x) = \hat{y} = wx + b$$

w and b are the parameters of the model

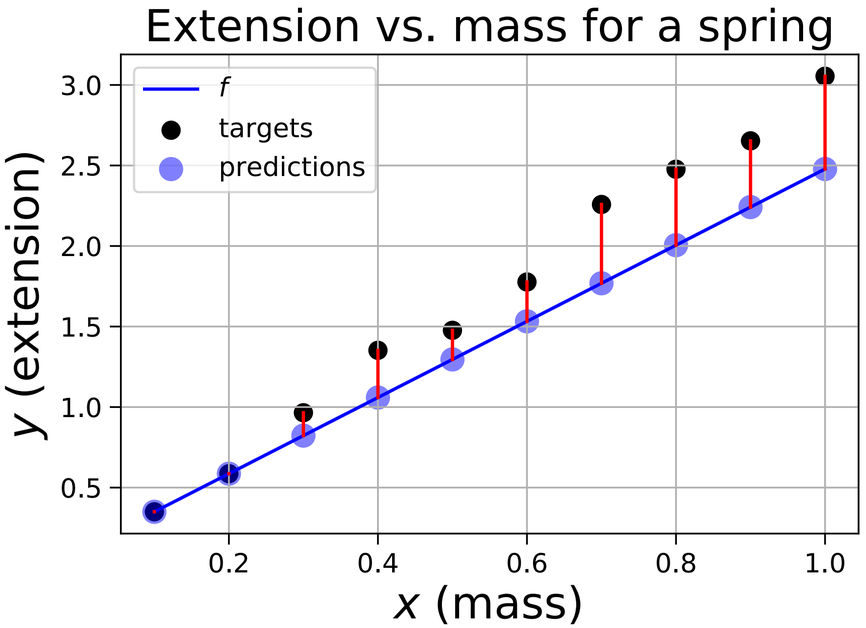
w is called the weight and b is called the bias



Our model predicts the targets

- $\hat{y}^{(1)}, \hat{y}^{(2)}, \ldots, \hat{y}^{(N)}$ are **predictions** of our **targets** $y^{(1)}, y^{(2)}, \ldots, y^{(N)}$
- We wanted a model f such that $\hat{y}^{(n)} = y^{(n)}$ for each point
- But we can't achieve this: a line can't perfectly fit the data here
- Can we relax our objective?

$$f(x) = \hat{y} = wx + b$$

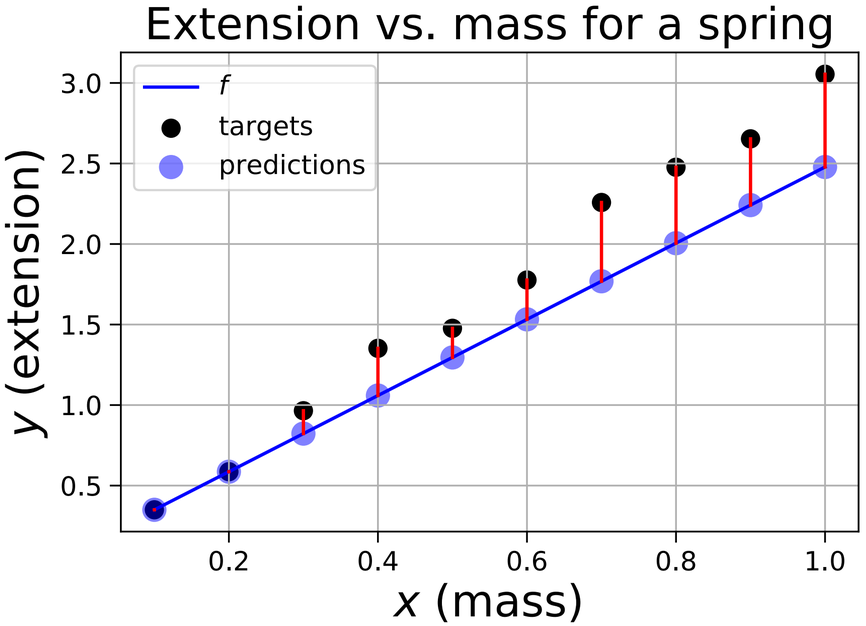


The squared error loss function

- Let's instead minimise the square distance between every $\hat{y}^{(n)}$ and $y^{(n)}$: $(y^{(n)} - \hat{y}^{(n)})^2$
- In ML, given an objective, we typically construct a loss function
- This is a function of the model parameters and the data

$$L_{SE} = \sum_{n} (y^{(n)} - \hat{y}^{(n)})^2$$

Our objective is achieved when the loss function is minimised



Minimising squared error

- Let's write $\mathbf{w} = [b \quad w]^\top$ and $\mathbf{x} = [1 \quad x]^\top$ so $f(\mathbf{x}) = \hat{y} = \mathbf{w}^\top \mathbf{x}$
- We want to find $\mathbf{w}$ that minimises $L_{SE}(\mathbf{w}) = \sum_n (y^{(n)} - \mathbf{w}^\top \mathbf{x}^{(n)})^2$
- We can express this loss as a vector norm with some rewriting:

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \vdots \\ y^{(N)} \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)\top} \\ \mathbf{x}^{(2)\top} \\ \mathbf{x}^{(3)\top} \\ \vdots \\ \mathbf{x}^{(N)\top} \end{bmatrix} \qquad L_{SE}(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$$

Vector calculus to the rescue

$$L_{SE}(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w})$$

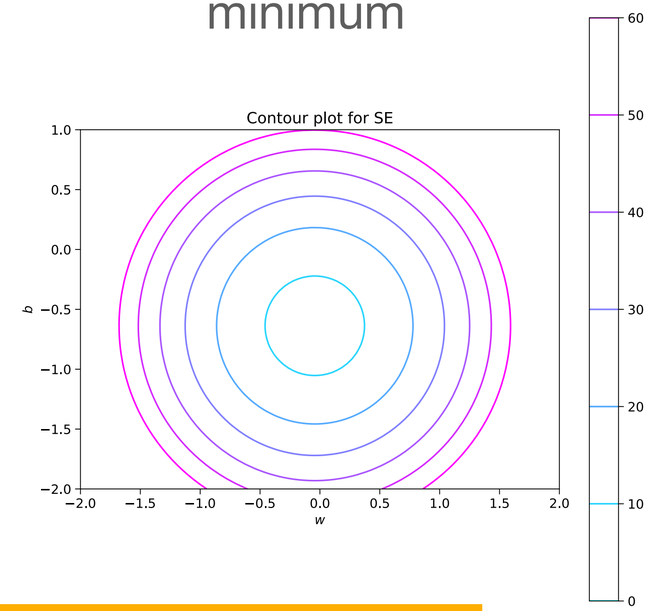
- Take the gradient and set to zero to get minimum

$$\nabla_{\mathbf{w}} L_{SE} = -2\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) = 0$$

- And rearrange

$$\mathbf{w}^* = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$$

This function is **convex:** it only has one extremum which is a minimum

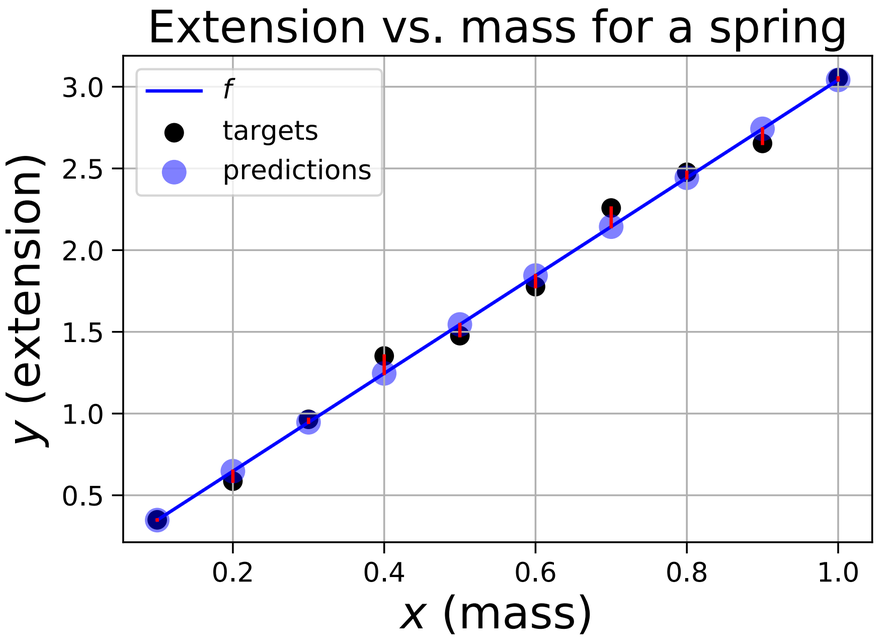


You are not required to do any vector or matrix calculus by hand on this course. https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf is a useful reference for this however.

A line of best fit

Compute $\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$ where $\mathbf{w}^* = [b^* \quad w^*]^\top$

This is the intercept and slope of a line that
minimises the distances
between target and predictions

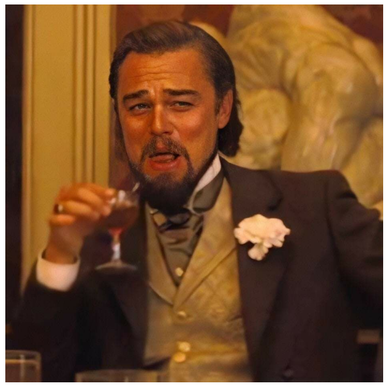


A probabilistic interpretation

- Let's make the perfectly ***normal*** assumption $p(y \mid \mathbf{x}) = \mathcal{N}(y; \mathbf{w}^\top \mathbf{x}, \sigma^2)$
- We would then want a model that maximises the probability of our targets across all our data points $\prod_n p(y^{(n)} \mid \mathbf{x}^{(n)})$ a.k.a. the **likelihood** of our data
- Maximising likelihood is the same as minimising negative log-likelihood
- After a bit of maths we can write the negative log-likelihood as:

$$\text{NLL}(\mathbf{w}) = \frac{1}{2\sigma^2} \sum_n (y^{(n)} - \hat{y}^{(n)})^2 + \frac{N}{2} \log(2\pi\sigma^2)$$

Minimising MSE loss is the same as maximising likelihood!



Multiple linear regression

- We just performed simple linear regression, mapping $\mathbb{R} \rightarrow \mathbb{R}$
- Multiple linear regression maps $\mathbb{R}^{D>1} \rightarrow \mathbb{R}$
- Let's predict petal width from the other three **features** in the iris dataset

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Our linear model is a weighted sum of the features plus a bias

Sepal length Sepal width Petal length

$$f(\mathbf{x}) = \hat{y} = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

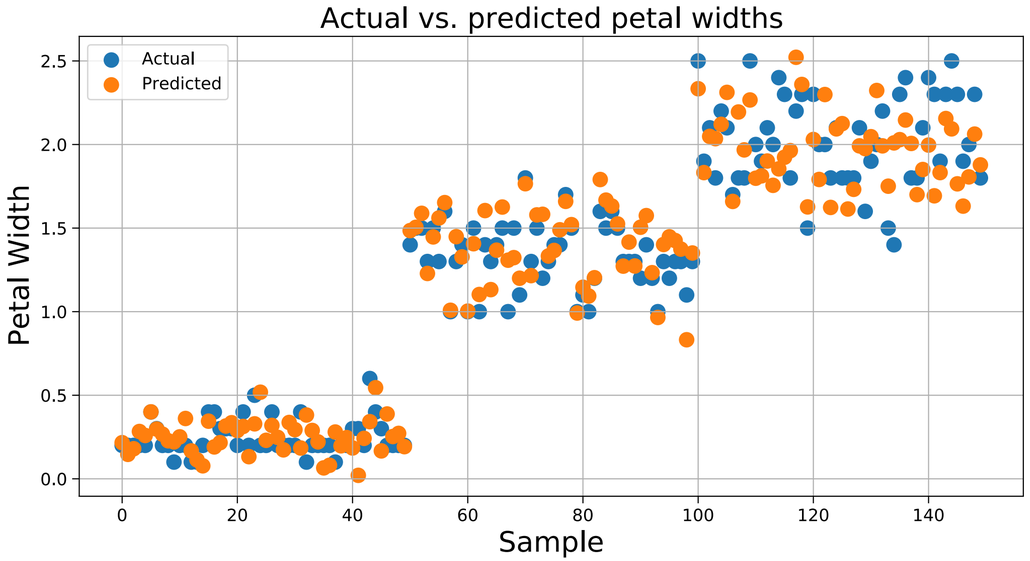
Petal width **prediction**

Minimising squared error (again!)

- Our **linear model** is $f(\mathbf{x}) = \hat{y} = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$
- We want to find the parameters w_1, w_2, w_3, b that minimise $L_{SE} = \sum_n (y^{(n)} - \hat{y}^{(n)})^2$
- Let's write $\mathbf{w} = \begin{bmatrix} b & w_1 & w_2 & w_3 \end{bmatrix}^\top$ and $\mathbf{x} = \begin{bmatrix} 1 & x_1 & x_2 & x_3 \end{bmatrix}^\top$
- This gives us $f(\mathbf{x}) = \hat{y} = \mathbf{w}^\top \mathbf{x}$ again
- $L_{SE} = \sum_n (y^{(n)} - \hat{y}^{(n)})^2 = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$
- Same solution: $\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$

Model evaluation: Mean squared error

- We could look at a plot our predictions $\hat{y}^{(1)}, \hat{y}^{(2)}, \ldots, \hat{y}^{(N)}$ against the targets $y^{(1)}, y^{(2)}, \ldots, y^{(N)}$ but it's nice to summarise performance using a **score**
- That score could be the **mean squared error** $L_{MSE} = \frac{1}{N} \sum_{n} (y^{(n)} - \hat{y}^{(n)})^2$



MSE is the average of the distances between predictions and targets

Here MSE is 0.03586. Low is good

Warning! MSE depends on the scale of your data

Model evaluation: Coefficient of Determination R^2

- R^2 is the default score for regression in sklearn
- It is 1 minus the reduction in error when you use your model's prediction instead of the mean of the targets

$$R^2 = 1 - \frac{\sum_n (y^{(n)} - \hat{y}^{(n)})^2}{\sum_n (y^{(n)} - \bar{y})^2}$$

Mean y

- It is maximally 1 (which is best) and can be negative if your predictions are **worse** than using the target mean!
- It can be seen as a measure of how much of the variance in the targets is explained by the model

Machine Learning is…

"the study of algorithms that can learn from training data in order to make predictions **on new data**."

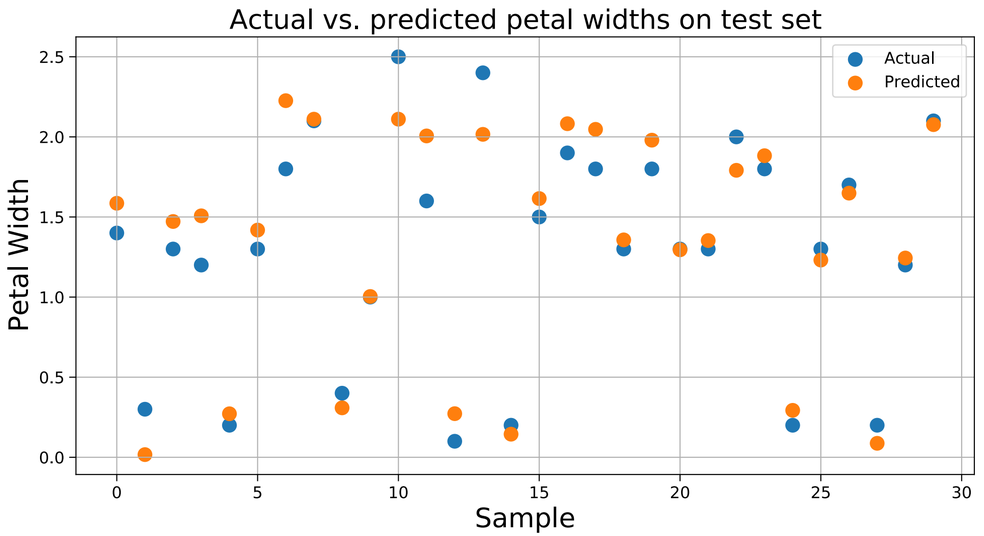
Elliot J. Crowley

Test set

- We ultimately want our model to do well on **new data**
- Models should be evaluated on data that **wasn't used for training**
- **Solution:** Evaluate model on a test set (can split dataset into train/test)
- A model that can perform well on test is able to **generalise**
- **The test set must never be used to fit the model**

Evaluation

- Let's split the iris dataset into 80% training and 20% test at random
- Learn weights on train, apply to test
- Train MSE: 0.03536 and Test MSE: 0.03906
- Train R^2: 0.9409 and Test R^2: 0.9179



How do we interpret the model?

$$\mathbf{w} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} -0.32 \\ -0.18 \\ 0.21 \\ 0.52 \end{bmatrix}$$

$$\hat{y} = -0.18x_1 + 0.21x_2 + 0.52x_3 - 0.32$$

Petal width **prediction** — Sepal length — Sepal width — Petal length

- With linear models, the weights tell you the contribution of each variable to the prediction
- **But this isn't simple to interpret if the data isn't standardised**

Variables have their own scales!

Standardised results

- We compute the variable means and standard deviations **on the training set**
- Then apply these **to the training set and the test set!**
- The learnt weights are now simple to interpret

$$\hat{y} = -0.15x_1 + 0.09x_2 + 0.92x_3 + 1.17$$

Petal width **prediction**

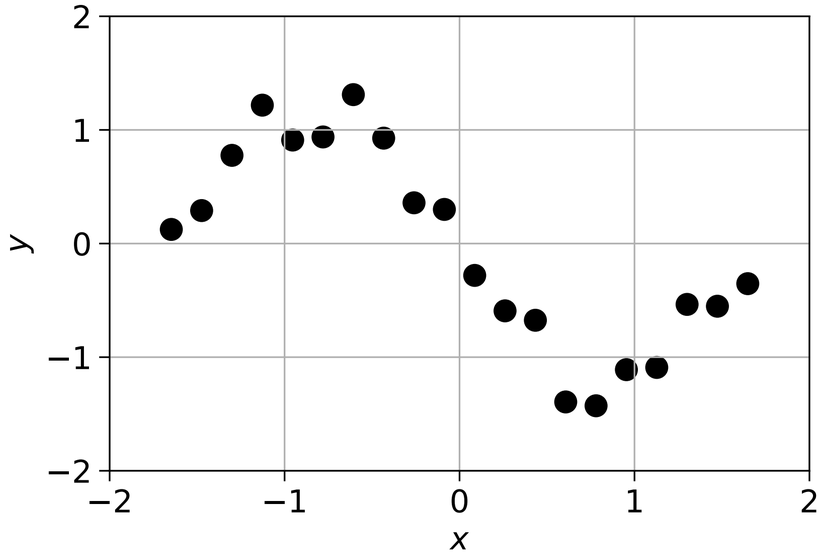
Standardised Sepal length

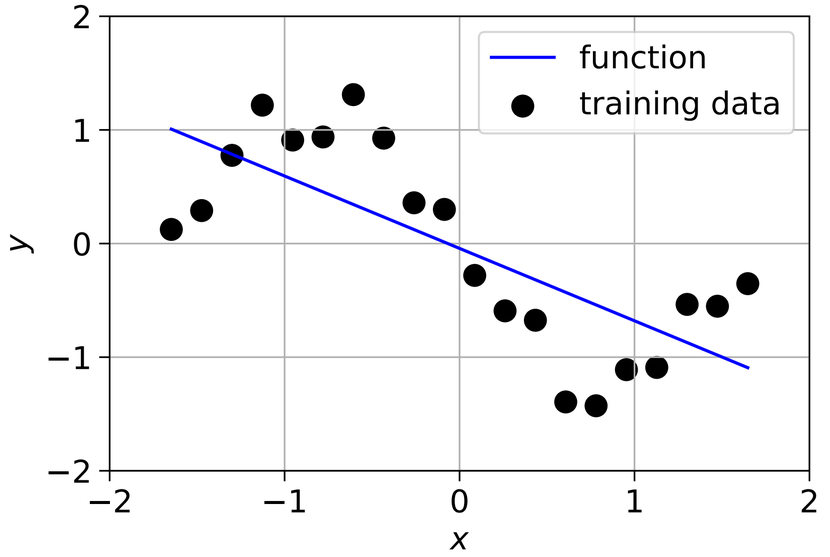
Standardised Sepal width

Standardised Petal length

Polynomial regression

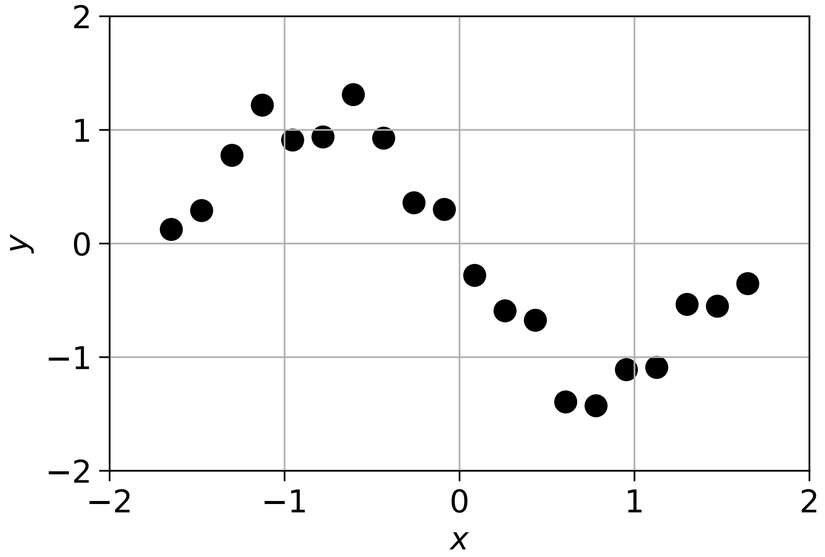
- Consider the 1D training set of data-target pairs below $\{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$
- The relationship between data and targets is curvilinear
- Simple linear regression produces a model that **underfits** to the data
- The model doesn't have the **capacity** to capture the way the data varies

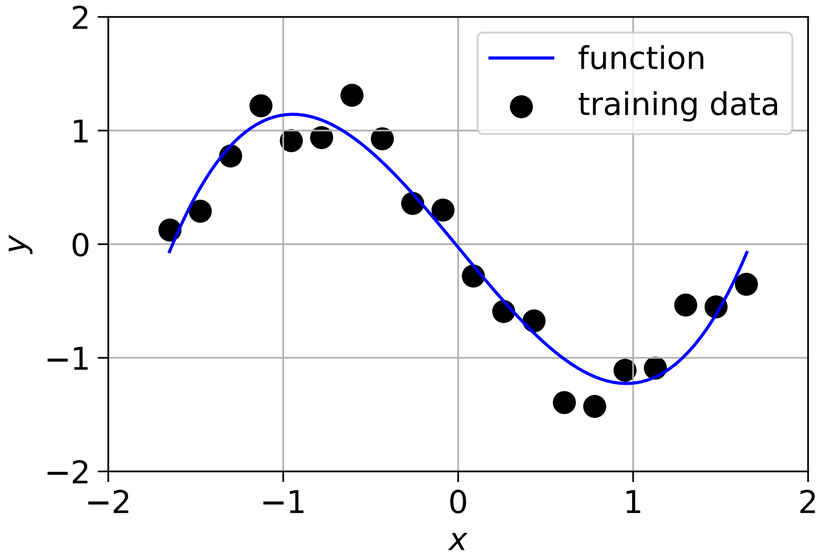




Polynomial regression

- Let's try fitting a polynomial $\hat{y} = f(x) = b + \sum_{m=1}^{M} w_m x^m$
- Using $M = 3$ we have $f(x) = b + w_1 x + w_2 x^2 + w_3 x^3$ and can get a good fit
- **The model is still linear in the weights**

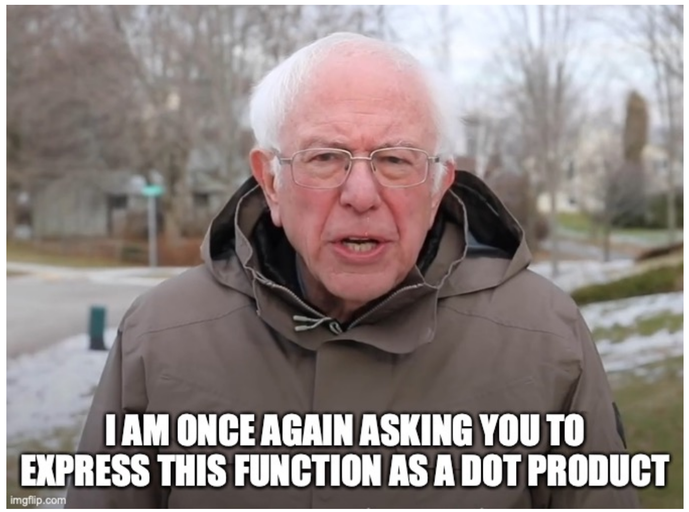




How do we fit this function?

- Our function is $\hat{y} = f(x) = b + \sum_{m=1}^{M} w_m x^m$

  1. Define $\phi(\mathbf{x}) = \begin{bmatrix} 1 & x & x^2 & \dots & x^M \end{bmatrix}^\top$
  2. Write $\mathbf{w} = \begin{bmatrix} b & w_1 & w_2 & \dots & w^M \end{bmatrix}^\top$



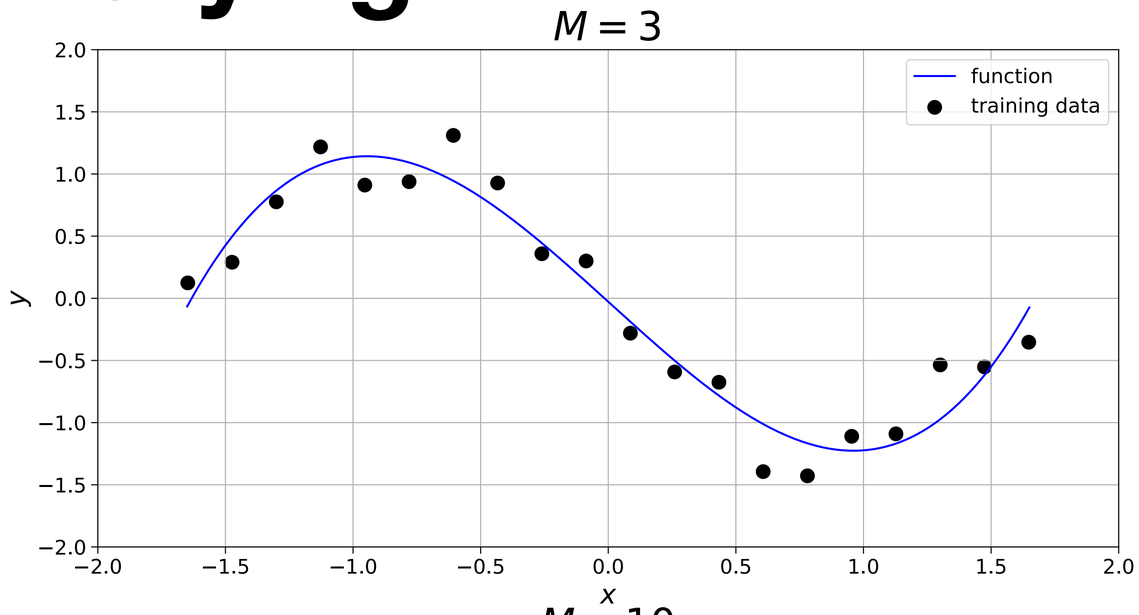
- We get $f(\mathbf{x}) = \hat{y} = \mathbf{w}^\top \phi(\mathbf{x})$. This looks familiar…
- It's the same as before except we have a **feature transformation** $\phi(\mathbf{x})$

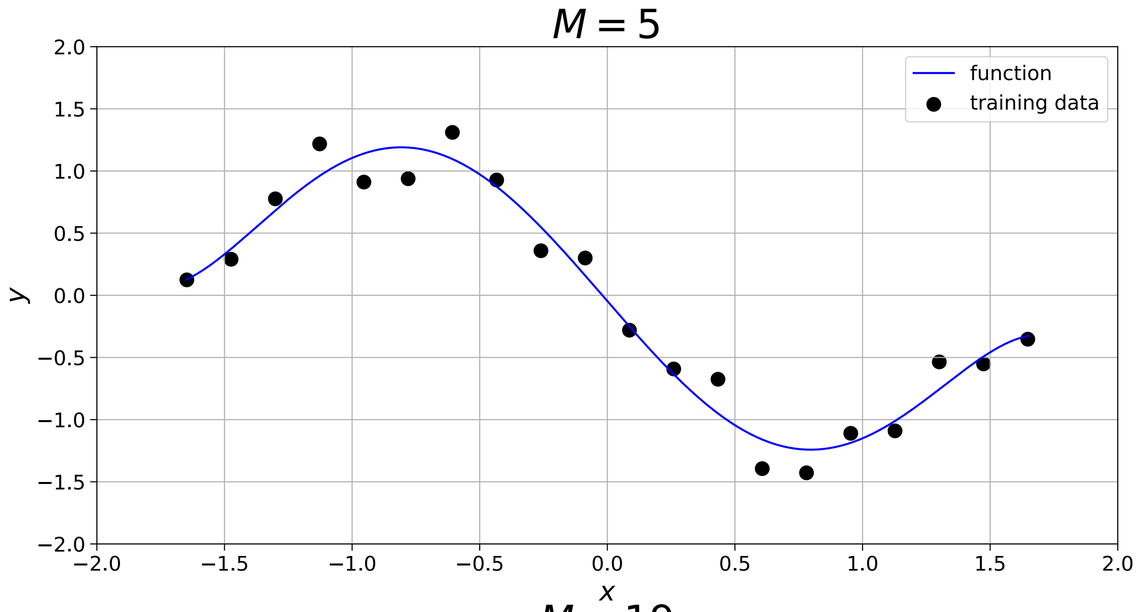
Minimising squared error (yet again)

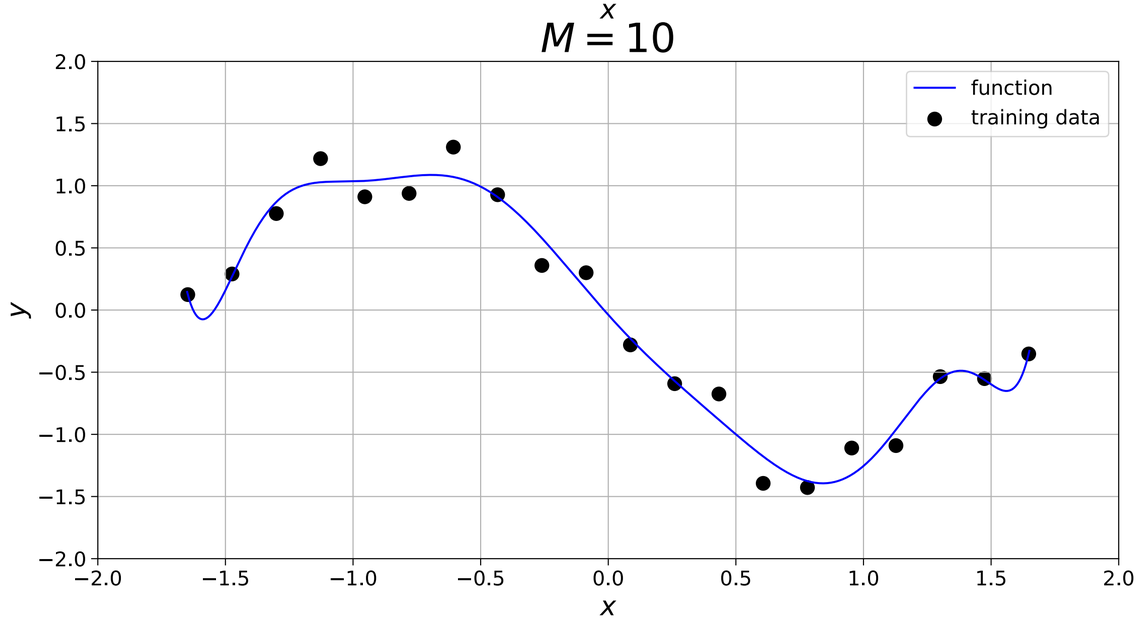
- Our **linear model** is $f(\mathbf{x}) = \hat{y} = \mathbf{w}^\top \phi(\mathbf{x})$
- We want to find the $\mathbf{w}$ that minimise $L_{SE} = \sum_n (y^{(n)} - \hat{y}^{(n)})^2$
- Define $\mathbf{\Phi} = \begin{bmatrix} \phi(\mathbf{x}^{(1)})^\top \\ \phi(\mathbf{x}^{(2)})^\top \\ \phi(\mathbf{x}^{(3)})^\top \\ \vdots \\ \phi(\mathbf{x}^{(N)})^\top) \end{bmatrix}$ then we get $L_{SE} = \sum_n (y^{(n)} - \hat{y}^{(n)})^2 = \|\mathbf{y} - \mathbf{\Phi}\mathbf{w}\|^2$
- Extremely similar solution: $\mathbf{w}^* = (\mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{y}$

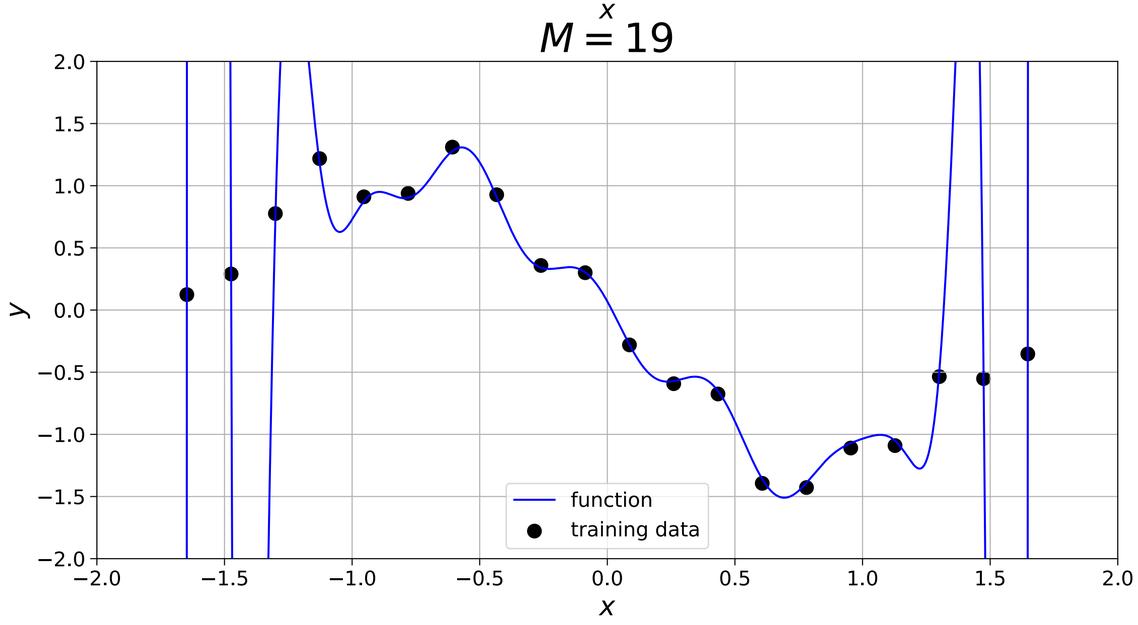
View $\mathbf{\Phi}$ and $\mathbf{X}$ as interchangeable here. It depends if we have a feature transformation or not

Varying M


$M = 3$
function
training data
x
y

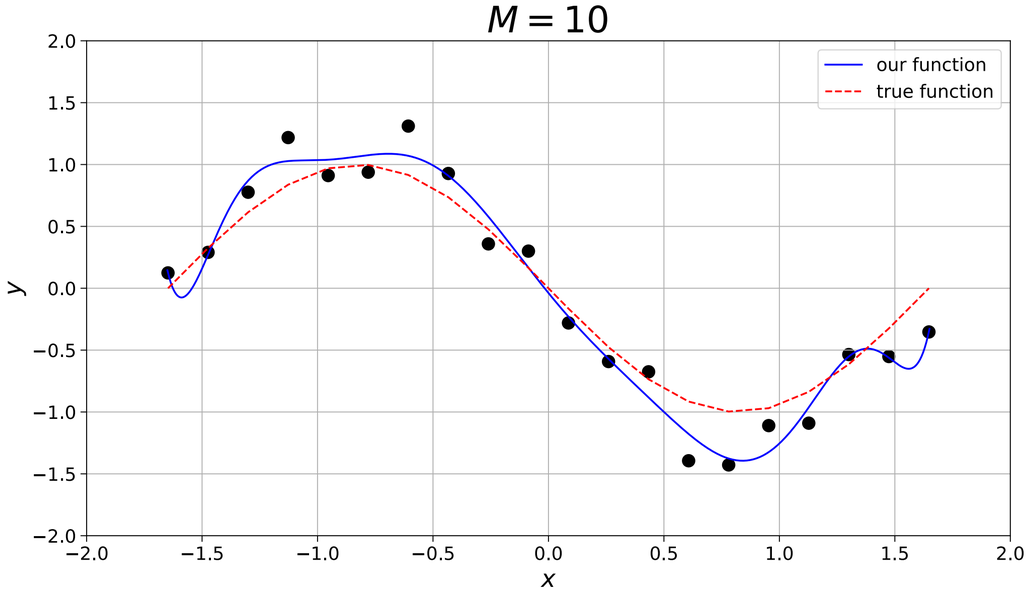

$M = 5$
function
training data
x
y

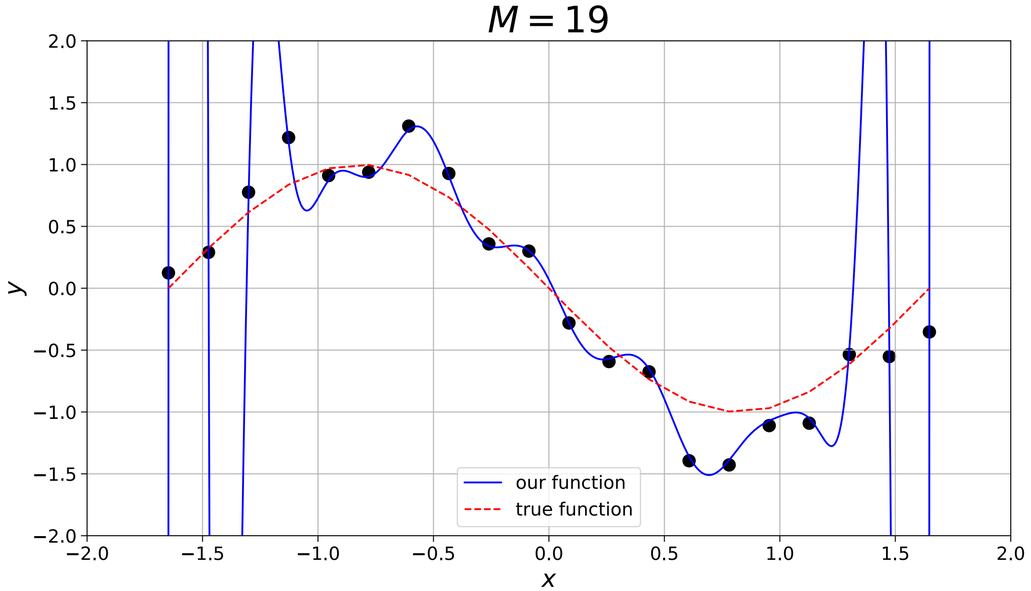

$M = 10$
function
training data
x
y


$M = 19$
function
training data
x
y

Overfitting

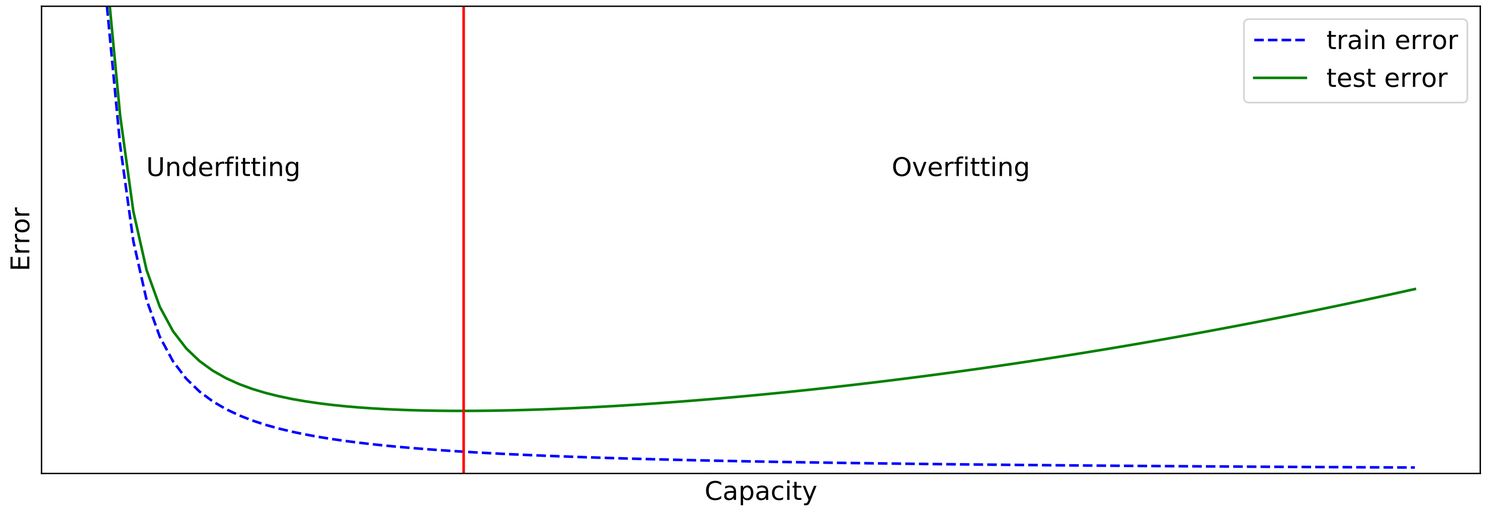
- These models have overfit to the training data
- We want our models to generalise to test data —these don't!
- **Spoilers**: $y = \sin((x - a)/b) + \mathcal{N}(0,0.25^2)$
- The models have too much **capacity,** and are latching on to the noise





Regularisation

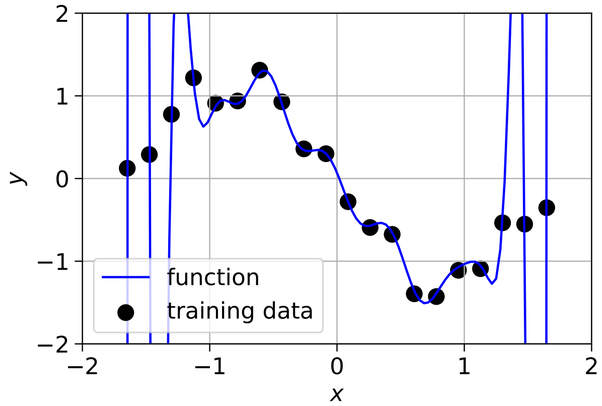
- We ultimately want to maximise test performance i.e. minimise test error
- The model should have the capacity to represent the function we care about
- But high capacity models tend to overfit
- **Regularisation** techniques combat overfitting by making the model *simpler*



This figure is my reproduction of Figure 5.3 from https://www.deeplearningbook.org/contents/ml.html

L2 regularisation

- Overfitted models tend to have large weights



$$y = 30.38x^{19} - 18.83x^{18} - 313.41x^{17} + \ldots$$

- We can regularise our model by penalising large weight values
- Let's add a term to our loss function that is small when weights are small

$$L_{ridge}(\mathbf{w}) = \underbrace{\|\mathbf{y} - \mathbf{\Phi w}\|^2}_{SE} + \underbrace{\lambda \|\mathbf{w}\|^2}_{regularisation}$$

Probabilistic interpretation: We are placing a Gaussian prior on the weights and performing MAP inference

Ridge regression

$$L_{ridge}(\mathbf{w}) = \underbrace{\|\mathbf{y} - \mathbf{\Phi w}\|^2}_{SE} + \underbrace{\lambda \|\mathbf{w}\|^2}_{regularisation} \quad \text{where } \|\mathbf{w}\|^2 = \mathbf{w}^\top \mathbf{w}$$

This function is **convex:** it only has one extremum which is a minimum

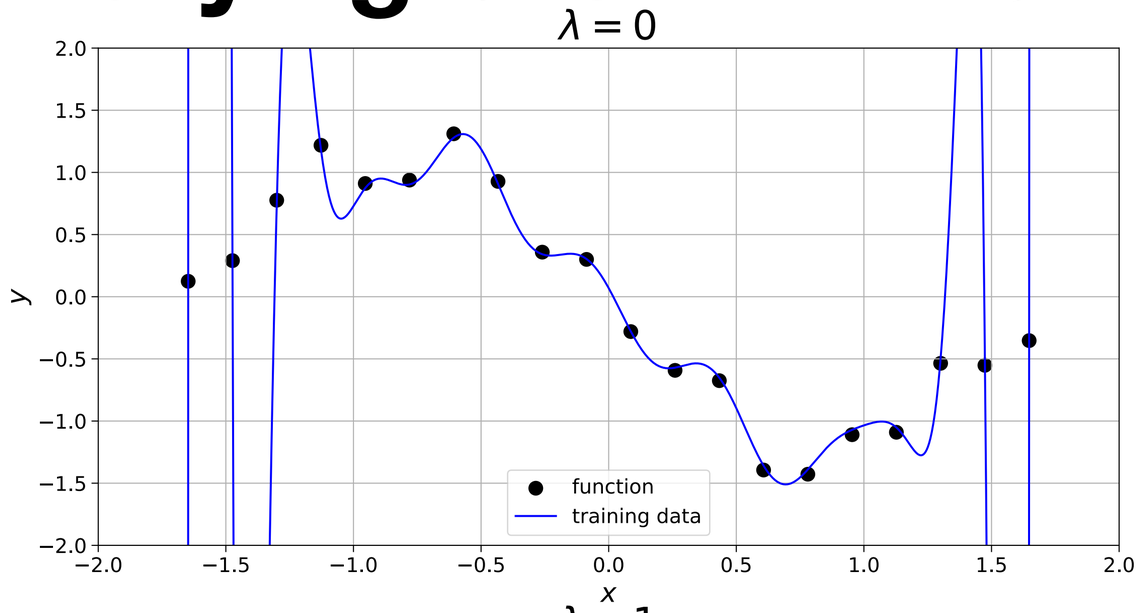
- λ is a hyperparameter that tells us how important regularisation is
- Let's take the gradient and set to zero to get the optimal weights

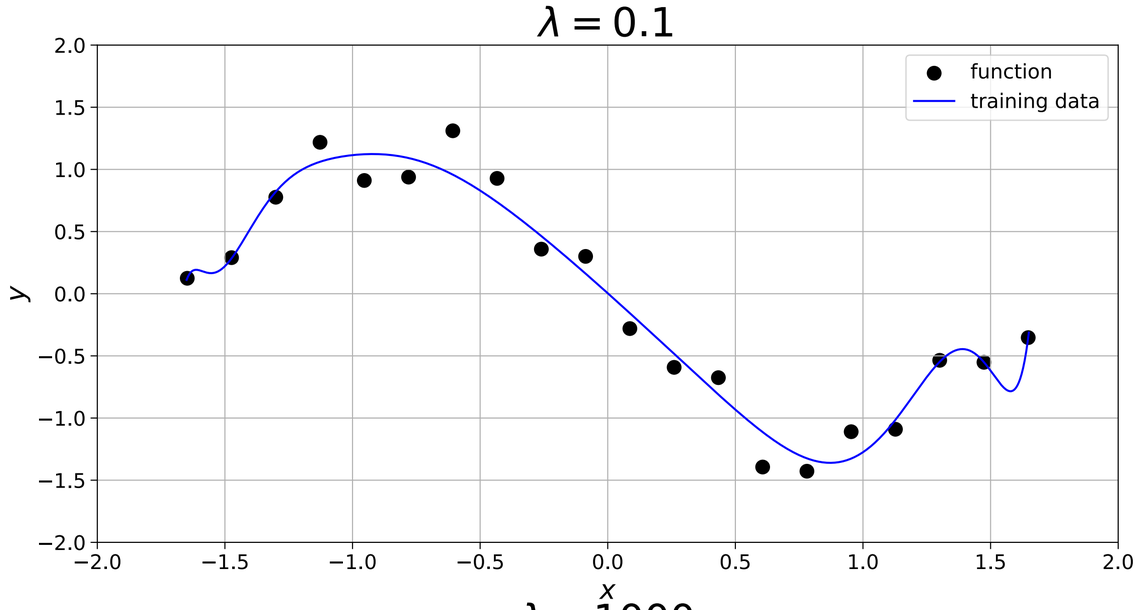
$$\nabla_{\mathbf{w}} L_{ridge} = -2\mathbf{\Phi}^\top(\mathbf{y} - \mathbf{\Phi w}) + 2\lambda \mathbf{w} = 0$$

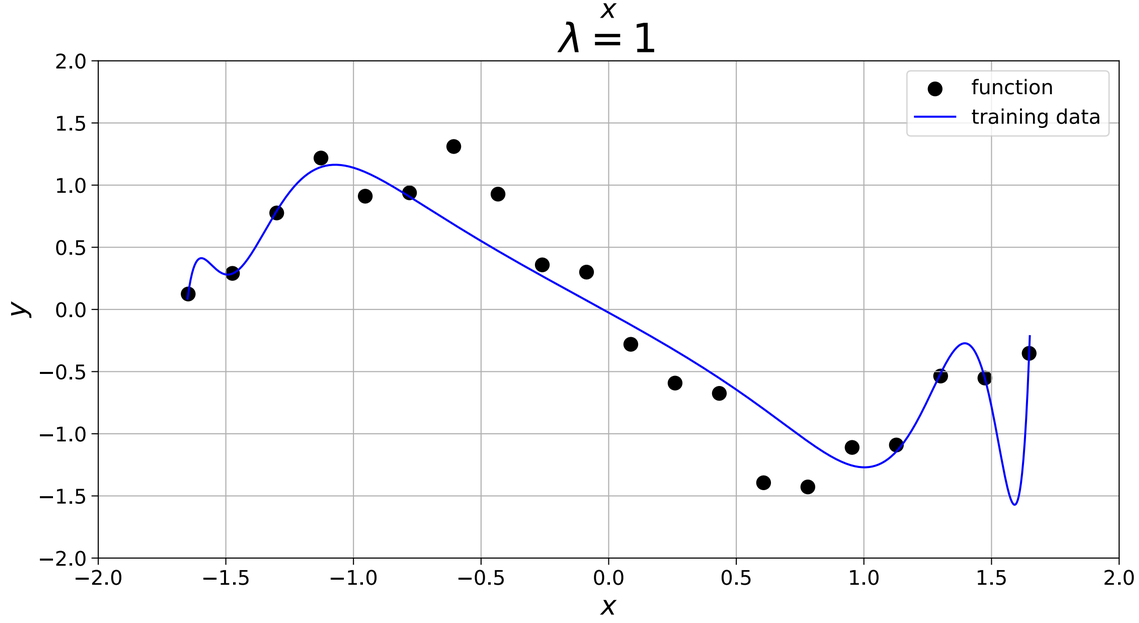
$$\mathbf{w}^* = (\mathbf{\Phi}^\top \mathbf{\Phi} + \lambda I)^{-1} \mathbf{\Phi}^\top \mathbf{y}$$

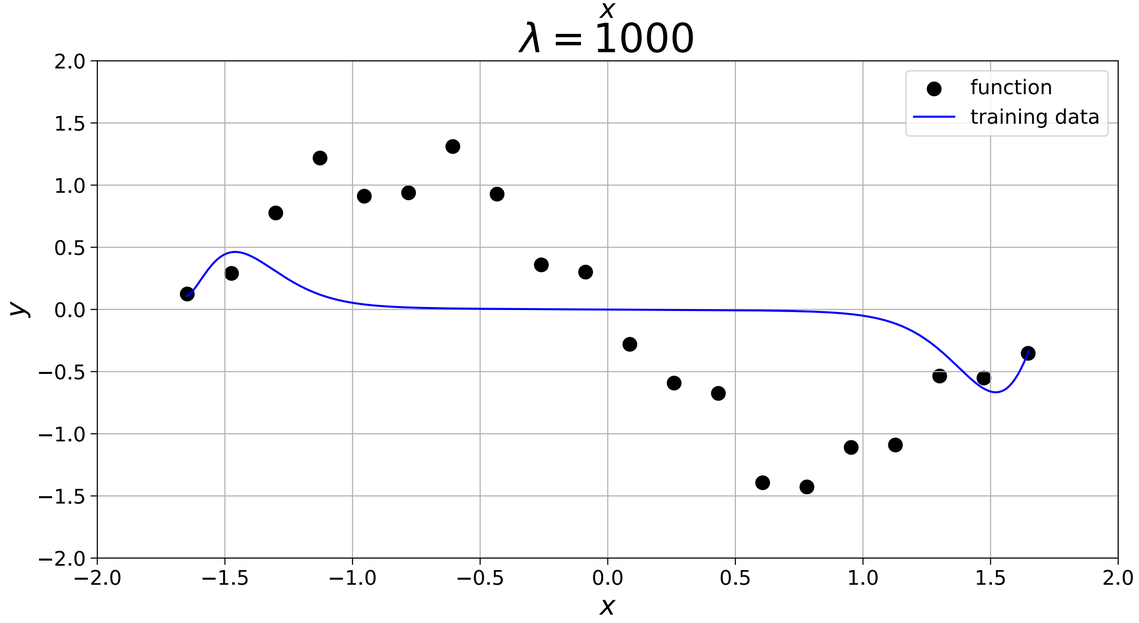
But in practice, we don't regularise the bias term. Sklearn will deal with this for you

Varying λ for $M = 19$









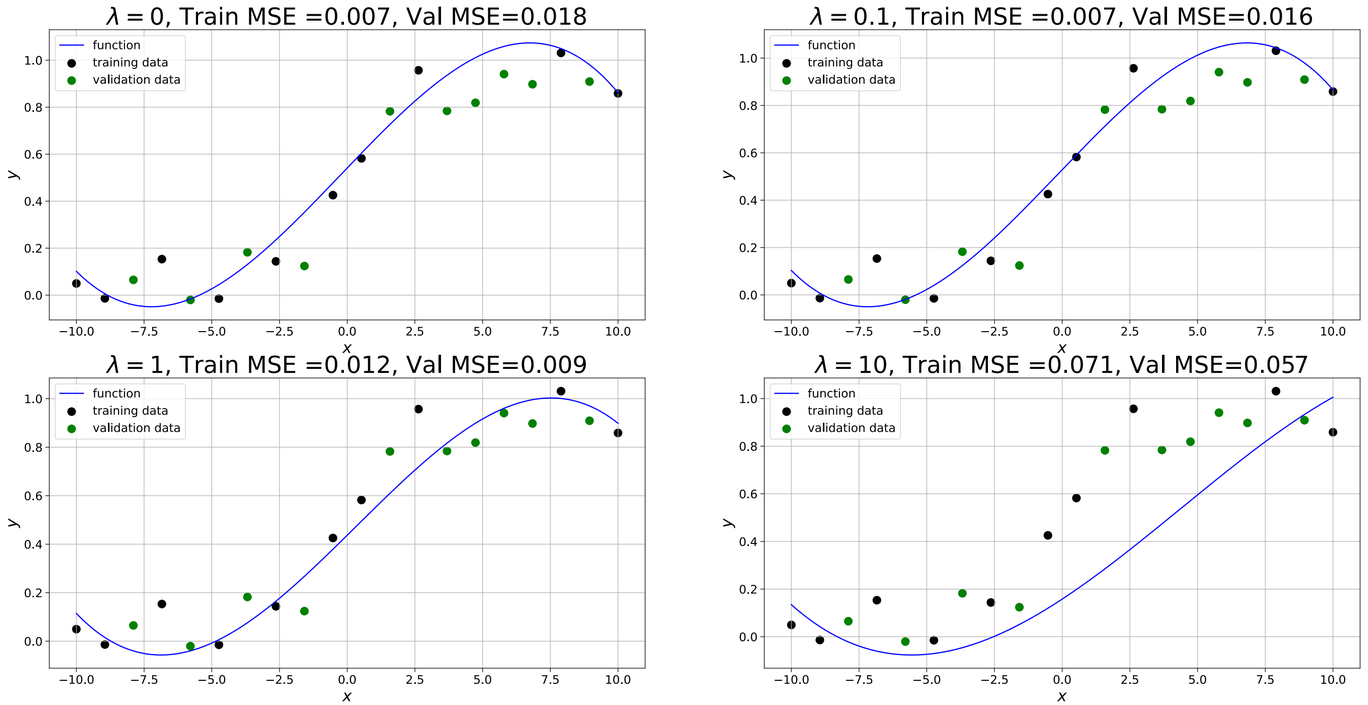
The validation set

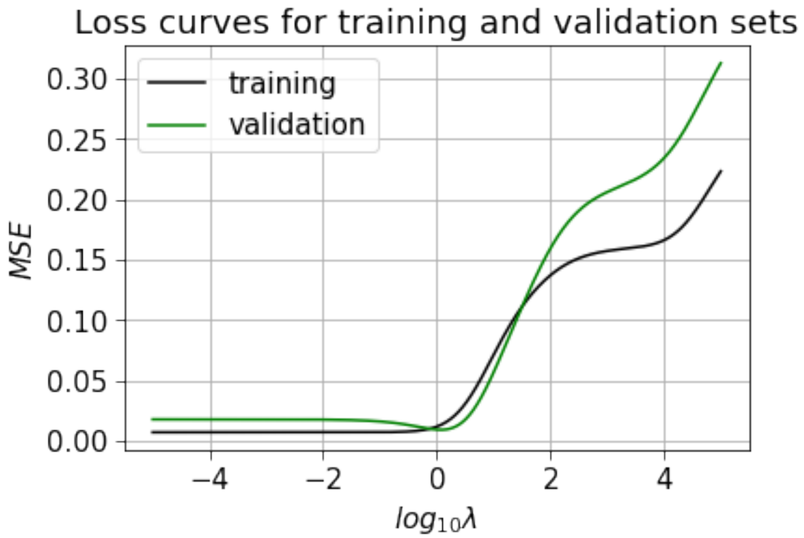
- Our goal is to perform well on the test set. Can we try different values of λ and pick the one that maximises test performance?
- **No! This would be using the test set to select the model**
- Instead, we split the dataset three-ways: train, validation, test
- The validation set is used for model selection
- i.e. we can evaluate models with different λ and **select** the one that does best on validation

Sometimes we will just use default hyperparameter values however

Hyperparameter tuning with grid search

- Create a list of λ values and for each value fit a model on the training set
- Evaluate each model on the validation set (e.g. with MSE or R^2)
- **Select** the model that performs best on validation **then** evaluate on test





Grid search

- We create a grid of possible values for each hyperparameter
- We then train a model for each grid element, and pick the model that performs best on the validation set. This is **model selection**
- With one hyperparameter, the grid is 1D, with two it's 2D and so on
- This can quickly get very expensive!

Imagine we have hyperparameters α and β. Let's search over $\alpha = \{0,1\}$ and $\beta = \{0.1,1,10\}$

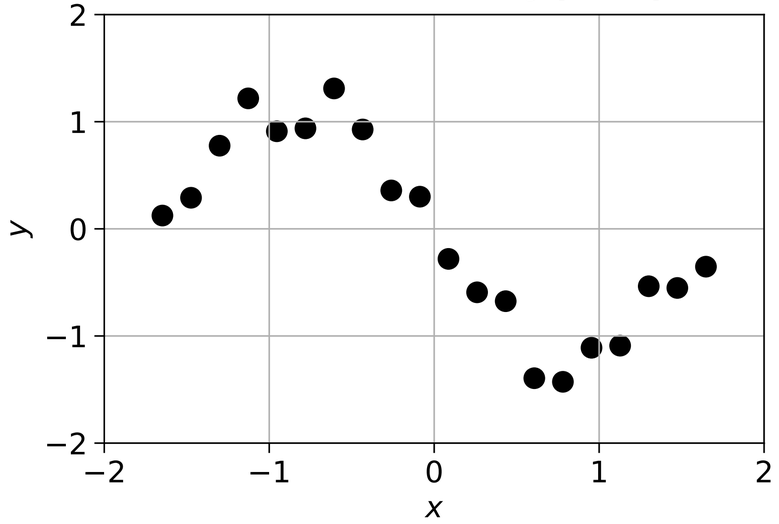
	$\beta = 0.1$	$\beta = 1$	$\beta = 10$
$\alpha = 0$	$R^2_{val} = 0.46$	$R^2_{val} = 0.52$	$R^2_{val} = 0.39$
$\alpha = 1$	$R^2_{val} = 0.73$	$R^2_{val} = 0.87$	$R^2_{val} = 0.79$

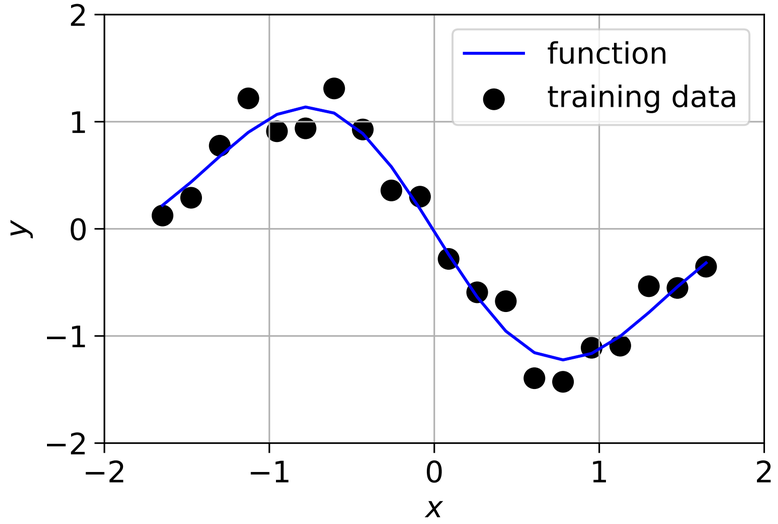
Other feature transformations are available

- We can design our own $\phi(\mathbf{x})$; $\mathbf{\Phi}$ is often referred to as the design matrix
- Each element could be a Gaussian centred on each training point

$$\phi(x) = \begin{bmatrix} e^{-(x-x^{(1)})^2/\sigma^2} & e^{-(x-x^{(1)})^2/\sigma^2} & e^{-(x-x^{(2)})^2/\sigma^2} & \dots & e^{-(x-x^{(N)})^2/\sigma^2} \end{bmatrix}^\top$$

- Here, σ is an additional hyperparameter





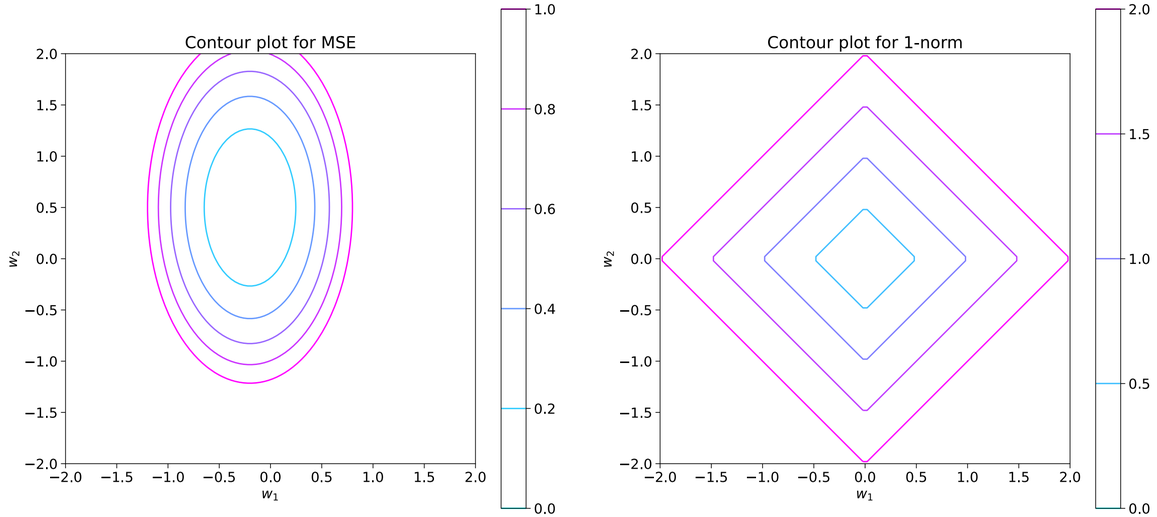
Lasso regression

$$L_{lasso}(\mathbf{w}) = \underbrace{\frac{1}{2N}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2}_{MSE} + \underbrace{\lambda\,|\,\mathbf{w}\,|}_{regularisation}$$

Remember that we can interchange $\mathbf{\Phi}$ and $\mathbf{X}$ in these equations

- Very similar to ridge regression except the SE term has been scaled and the regularisation term is a 1-norm

- 1-norm encourages sparsity in $\mathbf{w}$ which is a form of feature selection



The minimum occurs at one of the points where the contours of the two terms are at a tangent

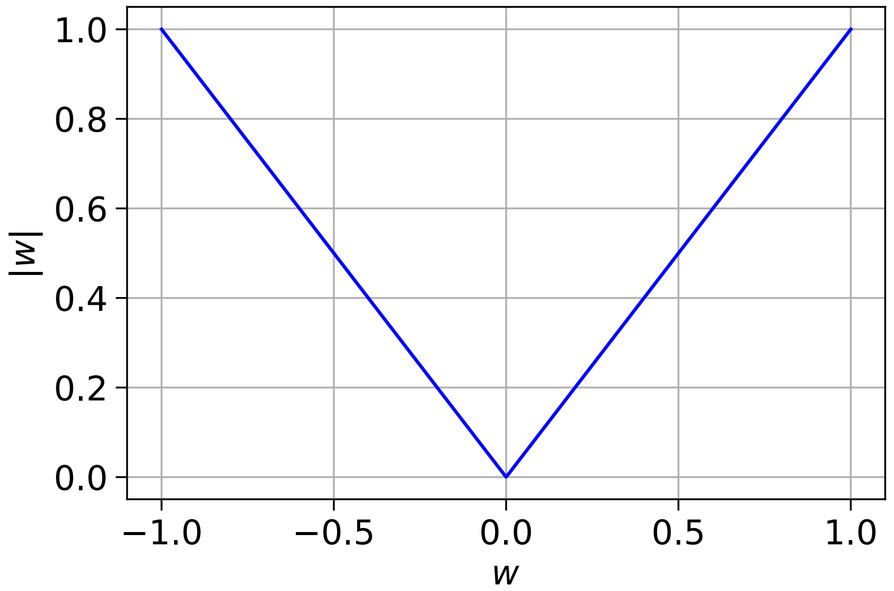
Such points are more likely to occur at the corners of the 1-norm contours

Probabilistic interpretation: We are placing a Laplace prior on our weights

Optimisation

- Finding the weights that minimise a loss function on training data is an optimisation problem $\underset{\mathbf{w}}{\text{minimise}}\ L(\mathbf{w})$ with solution $\mathbf{w}^* = \arg\min_{\mathbf{w}} L(\mathbf{w})$
- This was simple for $L_{ridge}(\mathbf{w})$ which is convex and differentiable
- We just compute $\nabla_{\mathbf{w}} L_{ridge}$ and set to zero
- However, $L_{lasso}(\mathbf{w})$ is non-differentiable

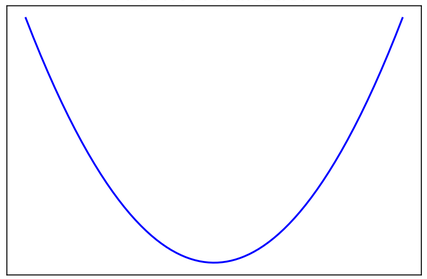
$$L_{lasso}(\mathbf{w}) = \underbrace{\frac{1}{2N}\|\mathbf{y} - \mathbf{\Phi w}\|^2}_{MSE} + \underbrace{\lambda\,|\,\mathbf{w}\,|}_{regularisation}$$



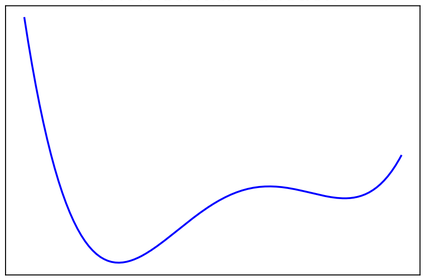
Convexity

- Convex functions have one extremum which is a minimum. This is very useful for optimisation!

A convex function

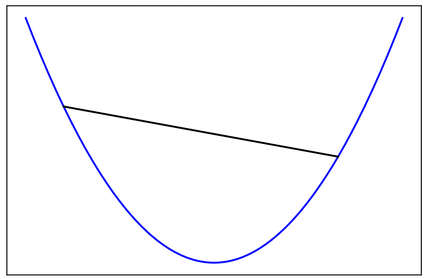


A non-convex function

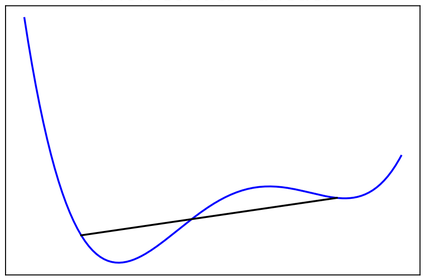


- A function of one variable is convex if a line drawn between any two points on the function doesn't fall below the function

A convex function

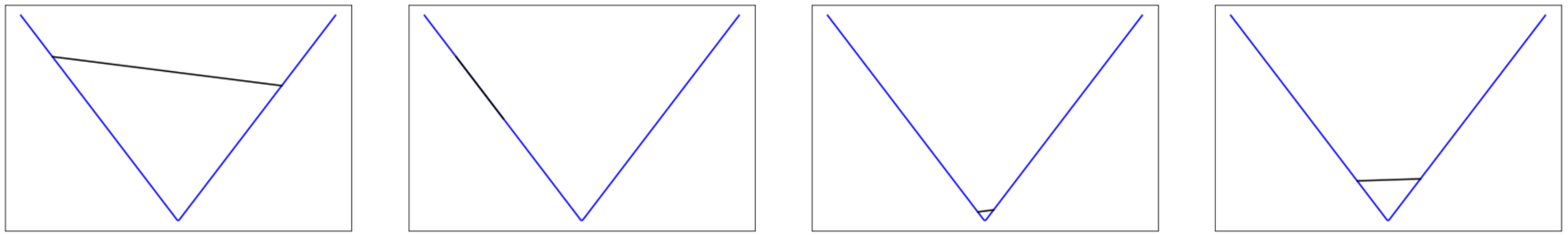


A non-convex function



Lasso is convex

- $|w|$ is convex (as is $|\mathbf{w}|$): it clearly has a minimum at $w = 0$.
- It not being differentiable doesn't change this



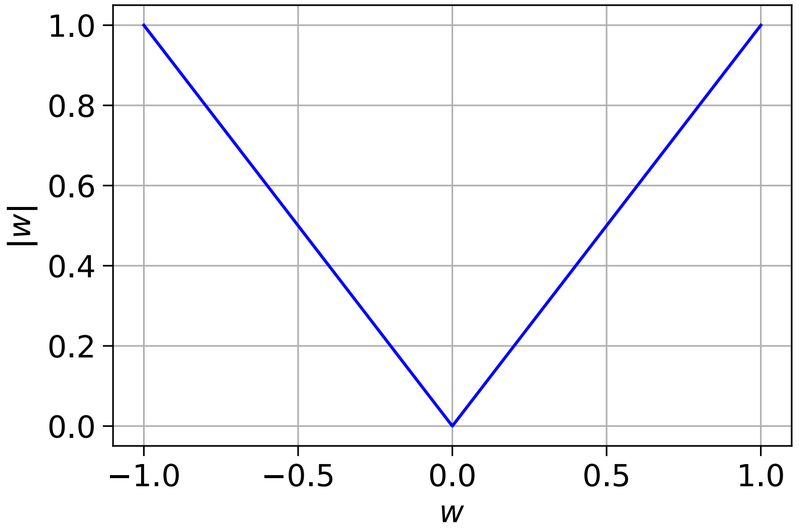
- The sum of two convex functions is convex

$$L_{lasso}(\mathbf{w}) = \underbrace{\frac{1}{2N}\|\mathbf{y} - \mathbf{\Phi w}\|^2}_{MSE} + \underbrace{\lambda|\mathbf{w}|}_{regularisation}$$

- $L_{lasso}(\mathbf{w})$ is convex, we just need to find its minimum

Subderivatives

- $g(w) = |w|$ is piecewise differentiable



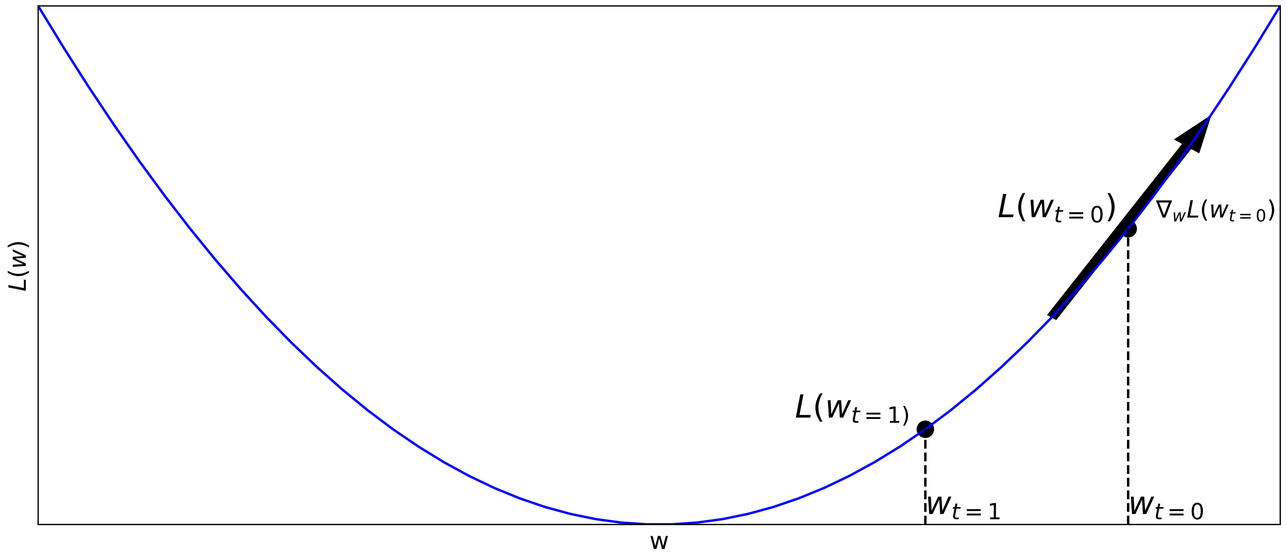
$$\frac{dg}{dw} = \begin{cases} 1 & \text{if } w > 0 \\ -1 & \text{if } w < 0 \end{cases}$$

- We can evaluate the gradient at any point (except $w = 0$)
- This is all we need to do to perform gradient descent (GD)

Gradient descent (GD) intuition

- We have a function $L(\mathbf{w})$ and we want to find $\mathbf{w}^* = \arg\min_{\mathbf{w}} L(\mathbf{w})$
- Let's initialise $\mathbf{w}$ at random and call it $\mathbf{w}_{t=0}$
- The gradient at $\mathbf{w}_{t=0}$: $\nabla_{\mathbf{w}} L(\mathbf{w}_{t=0})$ tells us **locally** the direction we can move $\mathbf{w}_{t=0}$ to most increase the function
- Move in the opposite direction!

$$\mathbf{w}_{t=1} = \mathbf{w}_{t=0} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w}_{t=0})$$



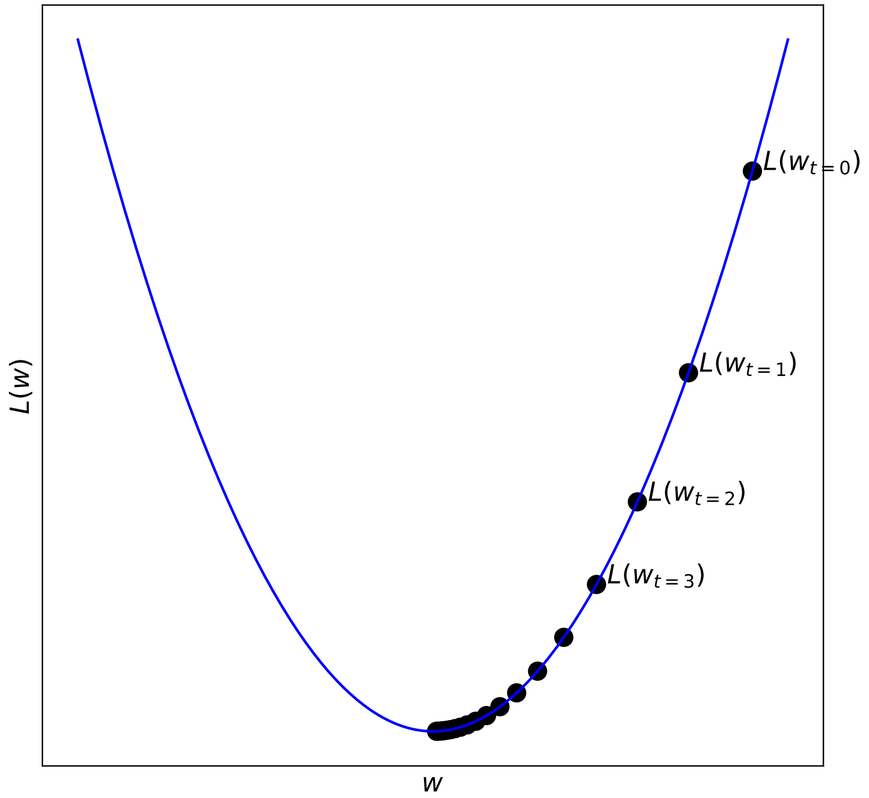
Gradient descent (GD) rationale

- Consider the loss at $\mathbf{w}_{t=i}$: $L(\mathbf{w}_{t=i})$
- Let's take a **small** step in weight space, so $\mathbf{w}_{t=i+1} = \mathbf{w}_{t=i} + \Delta$
- Because this is small we can approximate $L(\mathbf{w}_{t=i+1})$ using a 1st order Taylor expansion
- $L(\mathbf{w}_{t=i+1}) = L(\mathbf{w}_{t=i} + \Delta) \approx L(\mathbf{w}_{t=i}) + \nabla_{\mathbf{w}} L(\mathbf{w}_{t=i})^{\top} \Delta$
- If we set $\Delta = -\alpha \nabla_{\mathbf{w}} L(\mathbf{w}_{t=i})$ then $L(\mathbf{w}_{t=i+1}) \leq L(\mathbf{w}_{t=i})$ as long as α is small
- We can therefore keep taking steps to minimise loss

Gradient descent (GD) algorithm

Goal: We have a function $L(\mathbf{w})$ and we want to find $\mathbf{w}^* = \arg\min_{\mathbf{w}} L(\mathbf{w})$

- Initialise $\mathbf{w}$ as $\mathbf{w}_{t=0}$
- For i in range(T):
  1. Compute $\nabla_{\mathbf{w}} L(\mathbf{w}_{t=i})$
  2. Update $\mathbf{w}_{t=i+1} = \mathbf{w}_{t=i} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w}_{t=i})$



α is called the step size, or *learning rate*

It is yet another hyperparameter

Optimisation algorithms

- Using an optimisation algorithm to learn the weights that minimise a loss function on training data is known as **training** or **fitting** or **learning!**
- There are many optimisation algorithms; some work better than others for different methods
- We will only detail variations of gradient descent on this course
- Sklearn will default to whatever optimiser tends to work best for a method
- Please be happy using optimisation algorithms that you haven't learnt about, and if you're not — go find out how they work!

Summary

- We have learnt about linear regression
- We have reasoned about the need for a test set for evaluation
- We have discovered how regularisation can prevent overfitting
- We have learnt how a validation set can be used to perform model selection
- We have found out what convex functions are
- We have explored gradient descent for optimising convex functions