Boring but important disclaimers:

- If you are not getting this from the GitHub repository or the associated Canvas page (e.g. CourseHero, Chegg etc.), you are probably getting the substandard version of these slides Don't pay money for those, because you can get the most updated version for free at

  `https://github.com/julianmak/OCES4303_ML_ocean`

  The repository principally contains the compiled products rather than the source for size reasons.
- Associated Python code (as Jupyter notebooks mostly) will be held on the same repository. The source data however might be big, so I am going to be naughty and possibly just refer you to where you might get the data if that is the case (e.g. JRA-55 data). I know I should make properly reproducible binders etc., but I didn't...
- I do not claim the compiled products and/or code are completely mistake free (e.g. I know I don't write Pythonic code). Use the material however you like, but use it at your own risk.
- As said on the repository, I have tried to honestly use content that is self made, open source or explicitly open for fair use, and citations should be there. If however you are the copyright holder and you want the material taken down, please flag up the issue accordingly and I will happily try and swap out the relevant material.

OCES 4303 :

an introduction to **data-driven and ML methods** in ocean sciences

Session 2: recap of regression and probability, and more data handling

Outline

- supervised learning task as regression

  → recap of regression as an optimisation task

  → train/test split

  → measuring model skill

- data scaling

  → normalising to probability distribution function (pdf)

- (cross-)validation

  → questions of robustness in light of inherently randomness/uncertainties

  → overfitting and model hyperparameter tuning

  → k-fold cross-validation

Recap: Machine learning + regression

Recall regression is that, for X the input, Y the output, f the model, we want

$$y = f(X)$$

- prediction or forward problem: have X and f, want Y
- inference or inverse problem: have Y and f, want X
- inference/regression/Machine Learning: have X and Y, want f

Q. really want the "best" f, but how to define "best", and then obtain "best" f?

Recap: Linear regression

Recall linear regression takes the model as

$$\hat{Y} = aX + b$$

where $\hat{Y}$ is the prediction

- idea: given a measure of error $J = J(\hat{Y}, Y)$, find a and b such that J is minimised

  $\rightarrow$ i.e. we have an optimisation problem

Recap: Linear regression

Recall linear regression takes the model as

$$\hat{Y} = aX + b$$

where $\hat{Y}$ is the prediction

- idea: given a measure of error $J = J(\hat{Y}, Y)$, find a and b such that J is minimised

  $\rightarrow$ i.e. we have an optimisation problem

  $\rightarrow$ J is sometimes called the objective/mismatch/loss function (or functional depending on context)

  $\rightarrow$ a and b would be the model parameters of f, and control variables of the problem

!!! a and b would depend on choice of J

Recap: Linear regression

- one choice for J is the L^p family of norms

$$\|\hat{Y} - Y\|_{L^p} = \left(\int |\hat{Y} - Y|^p \, \mathrm{d}\mu \right)^{1/p}$$

$\rightarrow$ μ is a measure (but not going to elaborate what that is...)
$\rightarrow$ in practice we almost always deal with with sums instead of integrals...

- the two often encountered cases are L^2 as Mean Squared Error (MSE)

$$\mathrm{MSE} \sim \|\hat{Y} - Y\|_{L^2}^2 \sim \frac{1}{N} \sum_i^N |\hat{Y}_i - Y_i|^2$$

and L^1 as Mean Absolute Error (MAE)

$$\mathrm{MAE} \sim \|\hat{Y} - Y\|_{L^1} \sim \frac{1}{N} \sum_i^N |\hat{Y}_i - Y_i|.$$

Recap: Linear regression

- the standard linear regression uses L^2 as the loss function
  $\rightarrow$ finds the standard line of best fit (LOBF)
  $\rightarrow$ actually have close form solutions for a and b

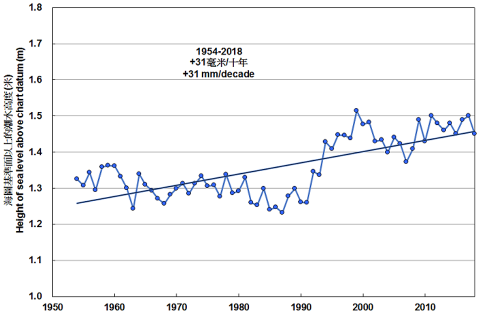

Figure: Figure from HKO.

- can go beyond linear
  $\rightarrow$ e.g. could argue figure on left is problematic...
- can also change choice of loss in principle
  $\rightarrow$ L^1 optimisation reduces effect of outliers

Train/test split + model skill

Good practice to not throw all the data in (overfitting etc.). Split (X, Y) into:

- training data $(X_{\text{train}}, Y_{\text{train}})$ (most data should be here)

  $\rightarrow$ exposed to ML algorithms for training the model

  $\rightarrow$ used to compute loss function
- test data $(X_{\text{test}}, Y_{\text{test}})$

  $\rightarrow$ **NOT** exposed to ML algorithm

  $\rightarrow$ used to test performance of model

Train/test split + model skill

Good practice to not throw all the data in (overfitting etc.). Split (X, Y) into:

- training data $(X_{\text{train}}, Y_{\text{train}})$ (most data should be here)
  $\rightarrow$ exposed to ML algorithms for training the model
  $\rightarrow$ used to compute loss function
- test data $(X_{\text{test}}, Y_{\text{test}})$
  $\rightarrow$ **NOT** exposed to ML algorithm
  $\rightarrow$ used to test performance of model

- sometimes have validation data $(X_{\text{val}}, Y_{\text{val}})$
  $\rightarrow$ subset of training data
  $\rightarrow$ exposed to ML algorithms to tune model hyperparameters and/or model selection

Train/test split

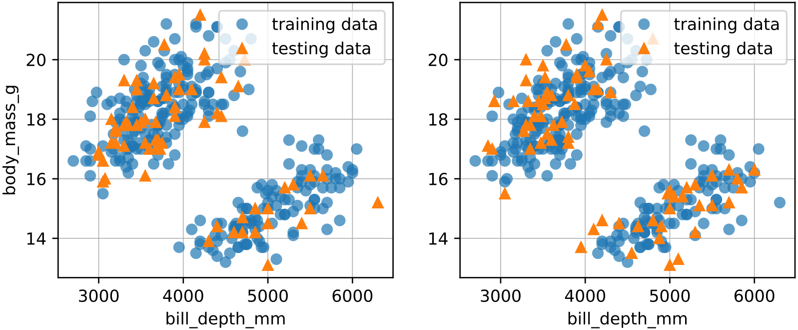

Figure: Sample of train/test split from penguins data.

Train/test split + model skill

- train/test split would be 'random'
  → i.e. sampling from a uniform distribution (see later)
  → introduces inherent randomness however (also see later)

- how to judge model skill on test data?
  → could use the L^p norms again
  → other choices, e.g. AIC and BIC that penalises complexity (see OCES 3301, and information/cross entropy later in course)

Ultimately there is a choice in J, and a lot of things boil down to how you choose that

(Lec 03 linear models is an exercise in varying J essentially...)

Data scaling

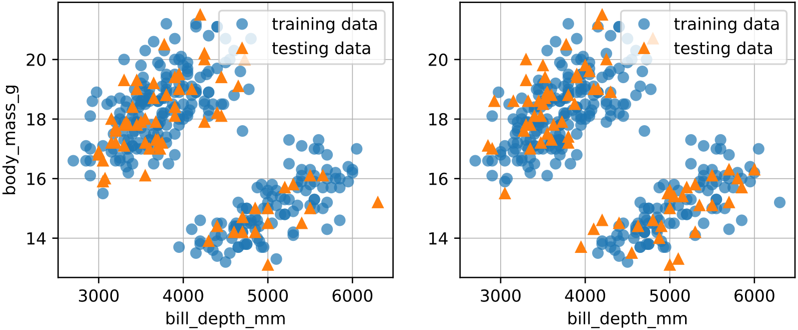

Figure: Sample of train/test split from penguins data.

- two variables here are different (e.g. units)

  $\rightarrow$ what to do about it?

  $\rightarrow$ how to compare different quantities in general? e.g. T and S and contribution to ρ

Recap: basics of probability

- idea: make the data distribution comparable
- recall that we say a random variable (just think data) X follows a Gaussian/normal distribution

$$X \sim \mathcal{N}(\mu, \sigma)$$

if it is described by the probability distribution function (pdf)

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

$\rightarrow$ just the 'bell' curve, with mean μ and variance σ^2

Recap: basics of probability

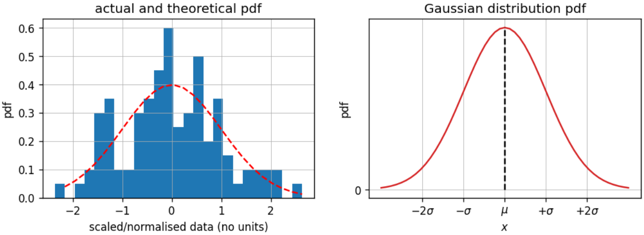

Figure: The Gaussian pdf (with units deliberately omitted). Obtain probability from an integral.

- CLT tells you with enough samples most distributions follow a Gaussian
- **68-95-99.7 rule**, 68, 95 and 99.7% of the data lies within 1, 2 and 3 std of the mean

Data scaling

- then for data $X \sim \mathcal{N}(\mu_X, \sigma_X)$ and $Y \sim \mathcal{N}(\mu_Y, \sigma_Y)$, just do Z-score standardisation (cf. OCES 3301 lec 05):

$$\tilde{X} = \frac{X - \mu_X}{\sigma_X}, \qquad \tilde{Y} = \frac{Y - \mu_Y}{\sigma_Y},$$

  then $\tilde{X}$ and $\tilde{Y}$ both follow $\mathcal{N}(0, 1)$

Data scaling

- then for data $X \sim \mathcal{N}(\mu_X, \sigma_X)$ and $Y \sim \mathcal{N}(\mu_Y, \sigma_Y)$, just do Z-score standardisation (cf. OCES 3301 lec 05):

$$\tilde{X} = \frac{X - \mu_X}{\sigma_X}, \qquad \tilde{Y} = \frac{Y - \mu_Y}{\sigma_Y},$$

then $\tilde{X}$ and $\tilde{Y}$ both follow $\mathcal{N}(0, 1)$

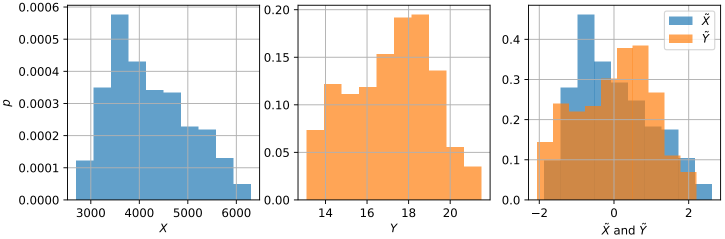

Figure: Raw and re-scaled pdfs as histograms.

Data scaling

- not the only way to do it, and not always the best way of doing it

  $\rightarrow$ e.g. not all data follows Gaussian distribution

  $\rightarrow$ e.g. sometimes there are principles to guide non-dimensionalisation

- if we don't have further information, could try this first

- aside: train/test split splits usually done via some uniform distribution, but can also do things differently

Validation and model robustness

- since we are talking about probability, there is inherent randomness

  → resulting models have some randomness also

  → how do we know our model 'works' not just because we got 'lucky'?

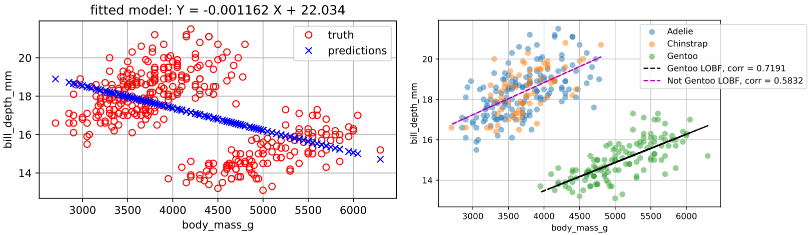

Figure: Contrived example of completely different models depending on data selection.

Validation and model robustness

- just do what we would normally do with statistics, e.g.

  $\rightarrow$ train an ensemble and take some averages

  $\rightarrow$ evaluate sensitivity to choices made

  $\rightarrow$ quantifying/probing for uncertainties

  $\rightarrow$ evaluate on possible over-fitting

  $\rightarrow$ ...

what you wouldn't do with statistics you should not with machine learning

Cross-validation

- over-fitting is when model has skill in training, but may suck in test

  → particularly if testing data is outside the pdf of the training data

- e.g. in OCES 3301 we did polynomial fitting

$$\hat{Y} = \sum_{i=0}^{p} a_i X^i = a_0 + a_1 X^1 + a_2 X^2 + \ldots$$

  → the degree of polynomial p could be regarded as the model hyperparameter here

  → blows up quickly outside of training range

- probably best to regarded most ML models as over-fitted unless shown otherwise...

e.g., k-fold cross validation

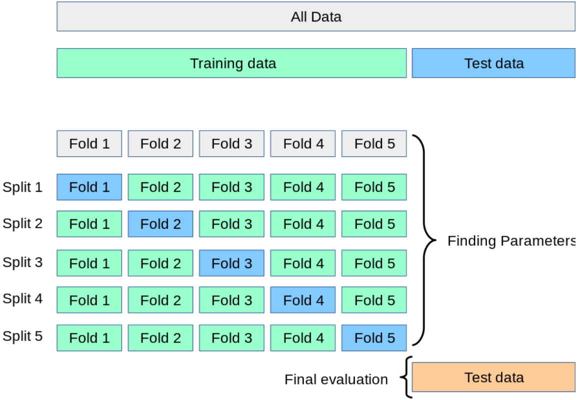

Figure: Schematic of k-fold cross-validation. Taken from `https://scikit-learn.org/stable/modules/cross_validation.html`.

- do repeated training on the folds, then do testing
  $\rightarrow$ the training thus also serve as validation set

Cross-validation

- then select 'best' model, e.g.
  $\rightarrow$ select best choice of p
  $\rightarrow$ select the $\{a_i\}$ that performs best during training
  $\rightarrow$ average the $(a_0, a_1, \ldots)$ and use the "averaged" model
  $\rightarrow \ldots$

- similarly if having a whole load of different models
  $\rightarrow$ for linear models next lecture, do the above but for linear regression, LASSO, ridge, etc.

Demonstration

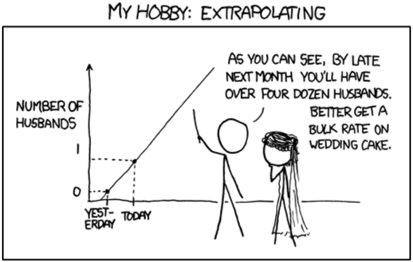

Figure: How not to do extrapolation. From XKCD.

what you wouldn't do with statistics you should not with machine learning

- Demonstrating most of above points using penguins data
  Moving to a Jupyter notebook →