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/OCES5303_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 5303 :
ML methods in Ocean Sciences

Session 2: more `sklearn`, cross-validation

Outline

- more `sklearn` things
  → linear models such as Linear Regression, Ridge, Lasso, Elastic-Net (cf. exercise in varing the loss function)
  → idea behind gradient descent
  → dimension reduction (e.g. PCA, lle, t-SNE)
  → clustering (e.g. K-means, others)

- cross validation and model hyper-parameter tuning
  → k-folds cross validation
  → ensemble methods

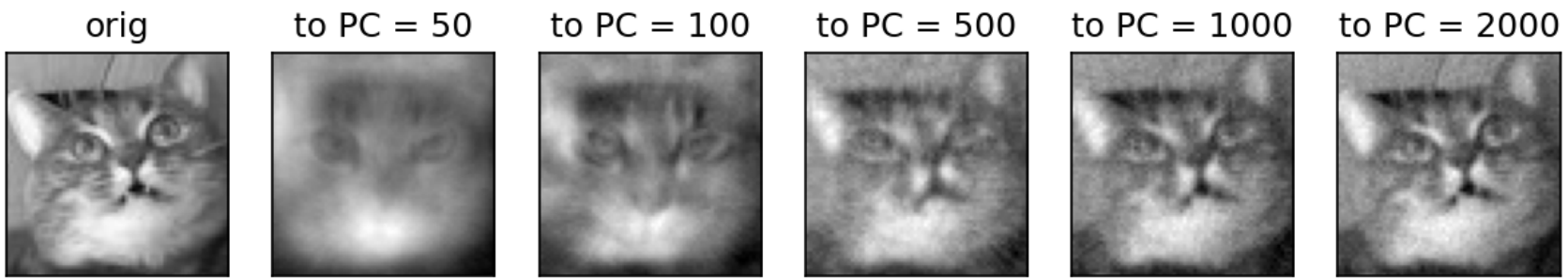


Figure: Result from an eigencat projection (it's really just PCAs).

Linear models

- given multiple features $X = (X^{(1)}, X^{(2)}, \ldots)$ and a **single** target Y, a linear model considers

$$\hat{Y} = a_0 + a_1 X^{(1)} + a_2 X^{(2)} + \ldots a_N X^{(N)}$$

  $\rightarrow$ the model parameters or control variables are $\{a_0, a_1, \ldots, a_N\}$

  $\rightarrow$ basically a linear algebra type problem (e.g. invert the matrices)

  $\rightarrow$ different methodologies take different loss functions J (see later)

- NOTE: this is not multi-variate regression where we have **multiple** targets Y, but see GLMs

Linear models: Linear Regression

- standard linear regression takes

$$J = \|\hat{Y} - Y\|_{L^2}^2 = \sum_{i=0}^{M} |\hat{Y}_i - Y_i|^2,$$

where Y_i is the training data and $\hat{Y}_i$ are the predictions from the model

$\rightarrow$ note lower index, M samples, but N lots of features described by $X^{(j)}$

$\rightarrow$ this basically provides the 1d LOBF embedded in $(N+1)$d space

$\rightarrow$ can have issues when features (the $X^{(j)}$) are not independent of each other

$\rightarrow$ can be sensitive to data noise

Linear models: Ridge

- consider controlling the size of the model coefficients a by

$$J = \|\hat{Y} - Y\|_{L^2}^2 + \alpha \|a\|_{L^2}^2$$

$\rightarrow$ this is peanlisation on a with strength α

- extra hyper-parameter α here

$\rightarrow$ large α means small coefficients and control colinearity, but at then expense of skill

- NOTE: different α's needed if data is not standardised

Linear models: Lasso

- could even limit number of features by asking for as many zero a_j as possible, via

$$J = \frac{1}{2M} \|\hat{Y} - Y\|_{L^2}^2 + \alpha \|a\|_{L^1}$$

$\rightarrow$ large α we want to keep as many non-zero coefficients as possible, but at then expense of skill

$\rightarrow$ relations to **compressed sensing**

Linear models: Elastic-nets

- could do a bit of both via

$$J = \frac{1}{2M}\|\hat{Y} - Y\|_{L^2}^2 + \alpha\rho\|a\|_{L^1} + \frac{\alpha(1-\rho)}{2}\|a\|_{L^2}^2$$

$\rightarrow \rho$ is a ratio hyper-parameter controlling sparsity and model parameter values

Sample loss function + gradient descent

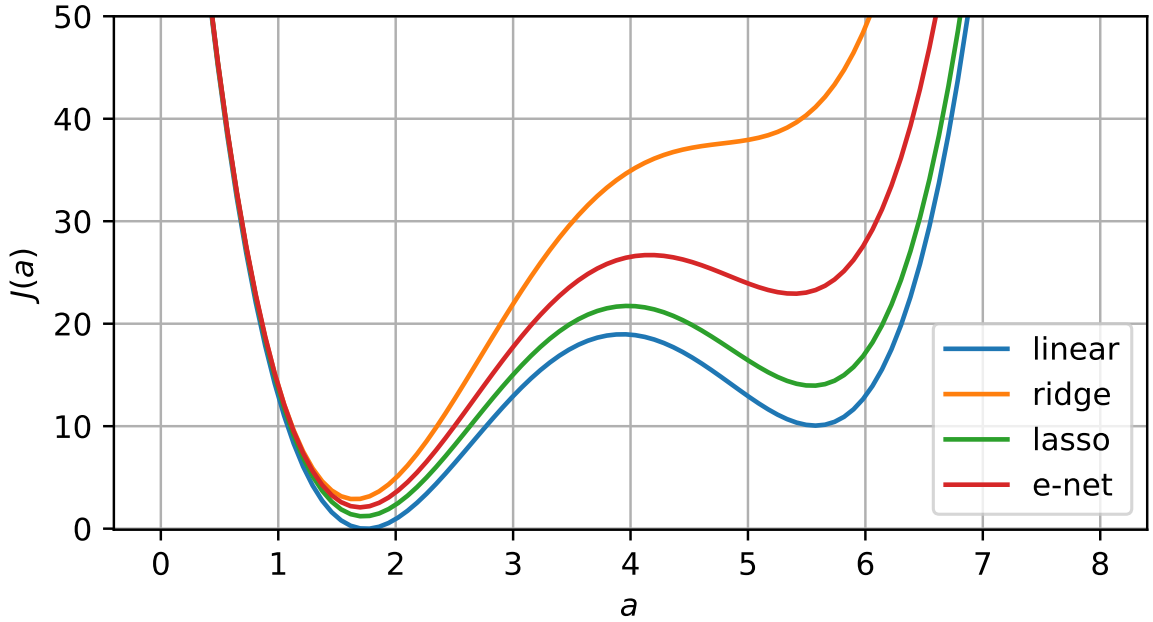


Figure: Made up 1d example of $J = J(a)$ with different choices of regularisation, $J = (1/2M)\|\hat{Y} - Y\|_{L^2}^2 + \alpha\rho\|a\|_{L^1} + (1/2)\alpha(1-\rho)\|a\|_{L^2}^2$.

- optimise by finding minimum, extremum occurs where the derivatives are zero $\Rightarrow$ root finding problem

  $\rightarrow$ **gradient** information is used to find direction of **descent**

Example: regressing to polynomial data

- going to do a demonstration case of taking

$$Y = X^2 + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

where σ controls the level of 'noise', but going to fit this to

$$\hat{Y} = a_0 + a_1 X^1 + a_2 X^2 + \ldots a_{10} X^{10}$$

i.e. my features $X^{(j)}$ in this case happens to polynomials X^j (X raised to power j)

$\rightarrow$ might expect to get $a_2 = 1$ and small values elsewhere

$\rightarrow$ a slightly more subtle meaning to "linear" is needed...

Example: regressing to polynomial data

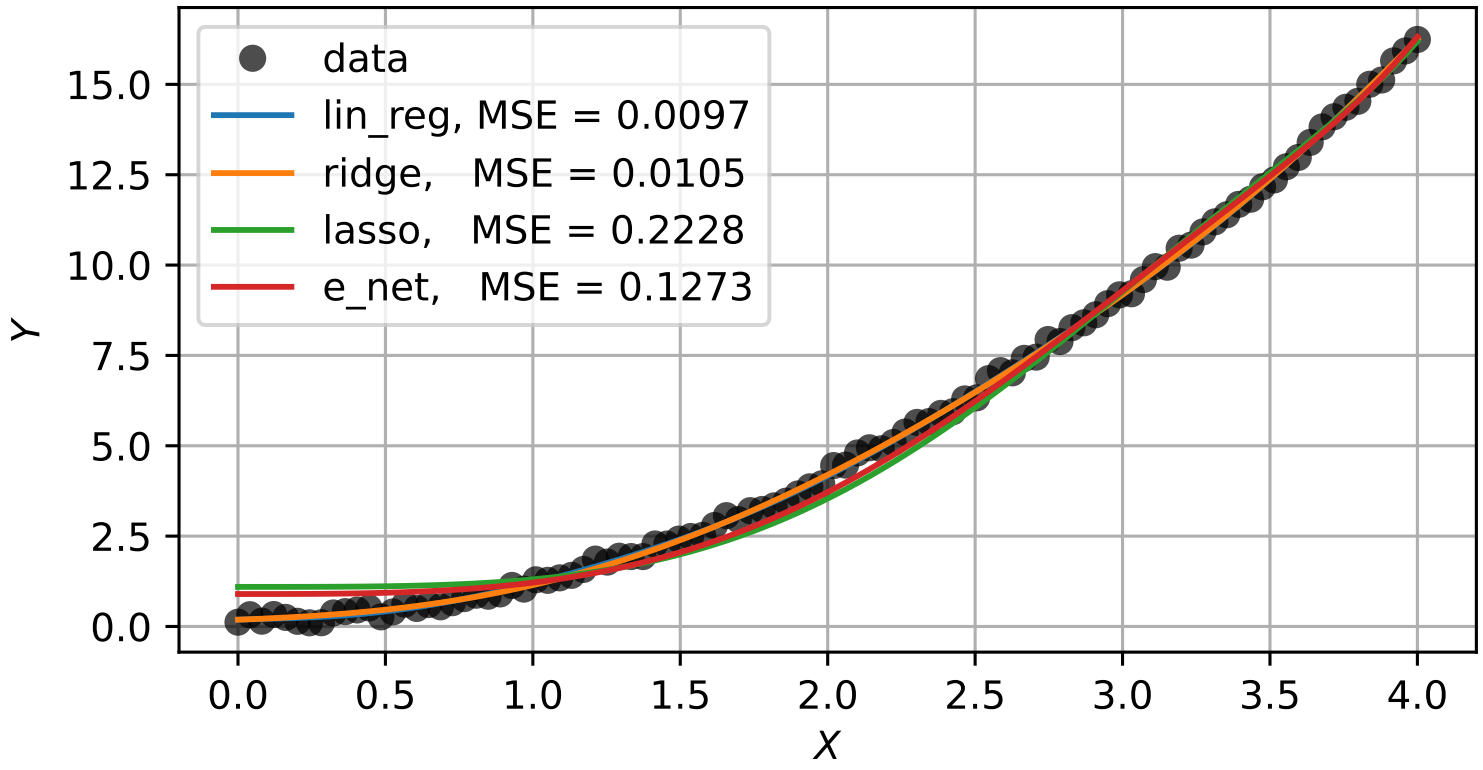


Figure: Polynomial regression for some contrived data, with MSE scores printed on.

Example: regressing to polynomial data

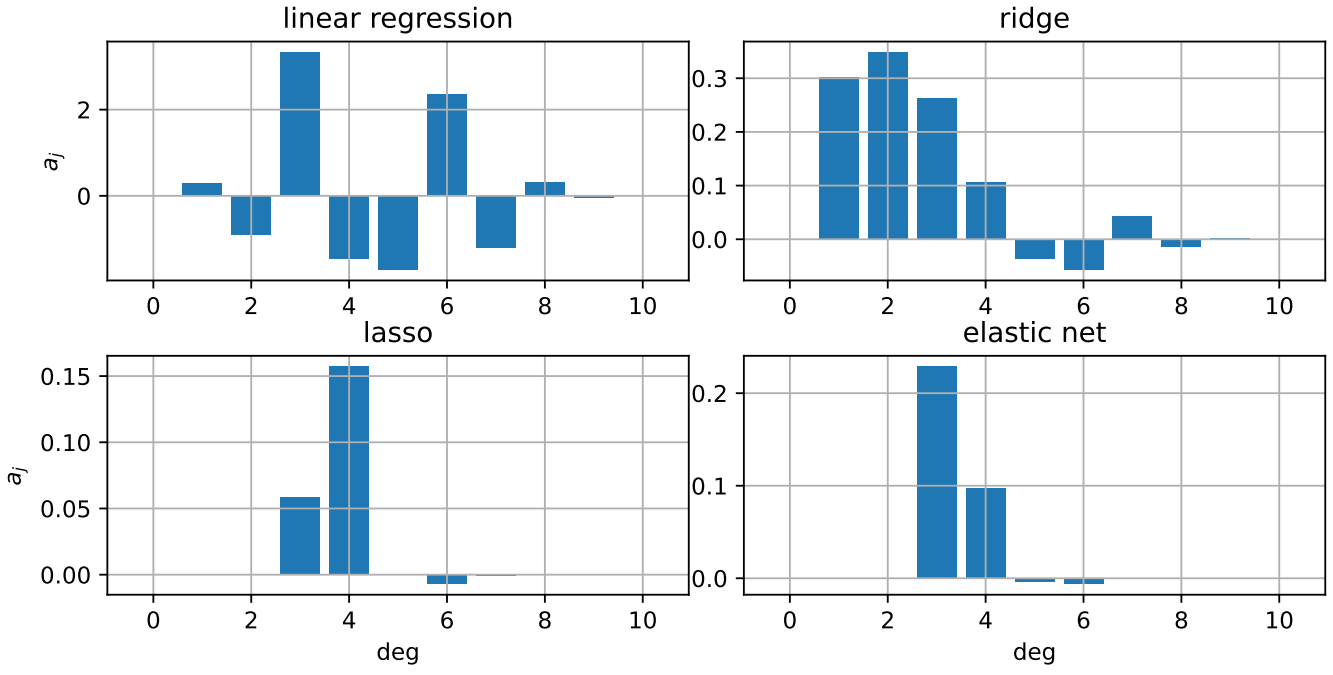


Figure: Polynomial regression for some contrived data, with MSE scores printed on.

- notice a_2 is not always largest!
- notice L^1 based penalisation (lasso and elastic net) gives more small coefficients

  $\rightarrow$ fewer features (more likely not over-fitted?)

Example: image classification + perceptrons (see Lec 04)

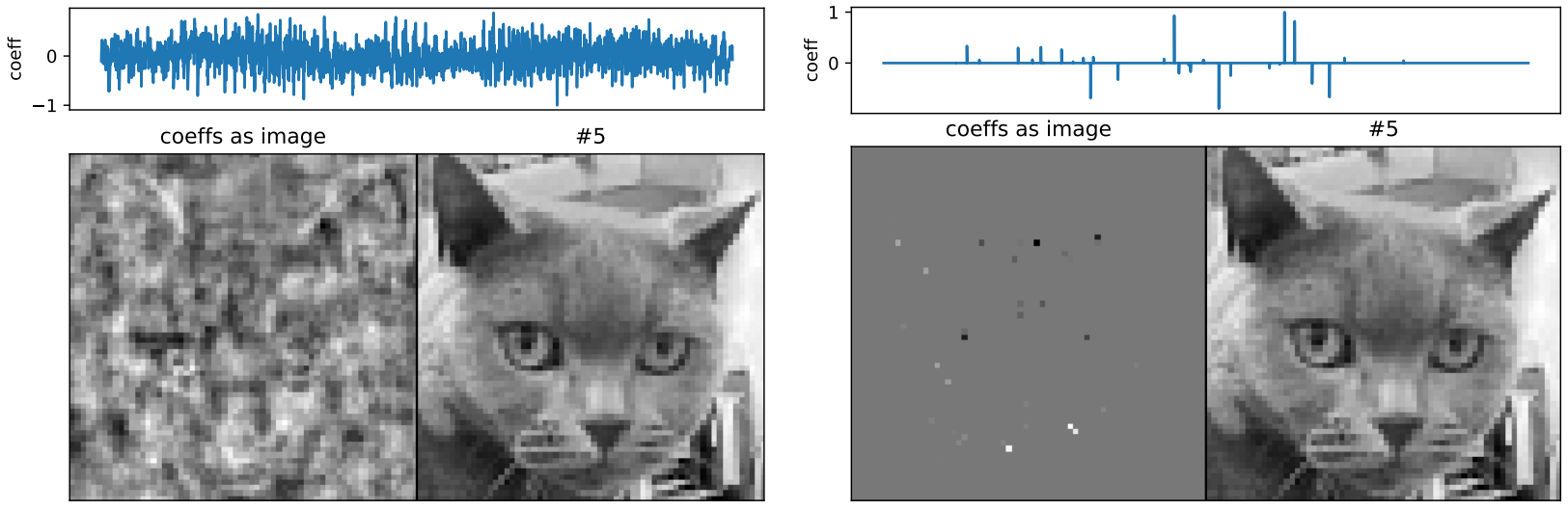


Figure: Visualisaing a perceptron from a pseudo-inversion of a matrix. (Left) Standard linear regression (L^2 inversion). (Right) Lasso (L^2 with a L^1 penalisation). See Lec 04.

- neural networks in its simplest form is just matrix inversion if the activation function is trivial

  $\rightarrow$ can obtain the model via a (pseudo-)inversion

  $\rightarrow$ basically an optimisation problem, can design loss functions accordingly

Dimension reduction: PCAs

Q. if not all features are as important, can find the important ones *a priori*?

$\rightarrow$ would help presumably against over-fitting

Dimension reduction: PCAs

Q. if not all features are as important, can find the important ones *a priori*?

$\rightarrow$ would help presumably against over-fitting

- Principal Component Analysis (PCA) is one way

  $\rightarrow$ shows up also as Empirical Orthogonal Functions (EOFs) particularly when pattern are involved
- considers linear combinations of features that explain the most variance in the data

  $\rightarrow$ effectively does a SVD, returns
  1. singular values σ (related to variance explained)
  2. the left and right singular vectors that gives the new co-ordinate system and relevant expansions

Dimension reduction: PCAs

- easier to explain visually with the penguins
  $\rightarrow$ data has been standardised here

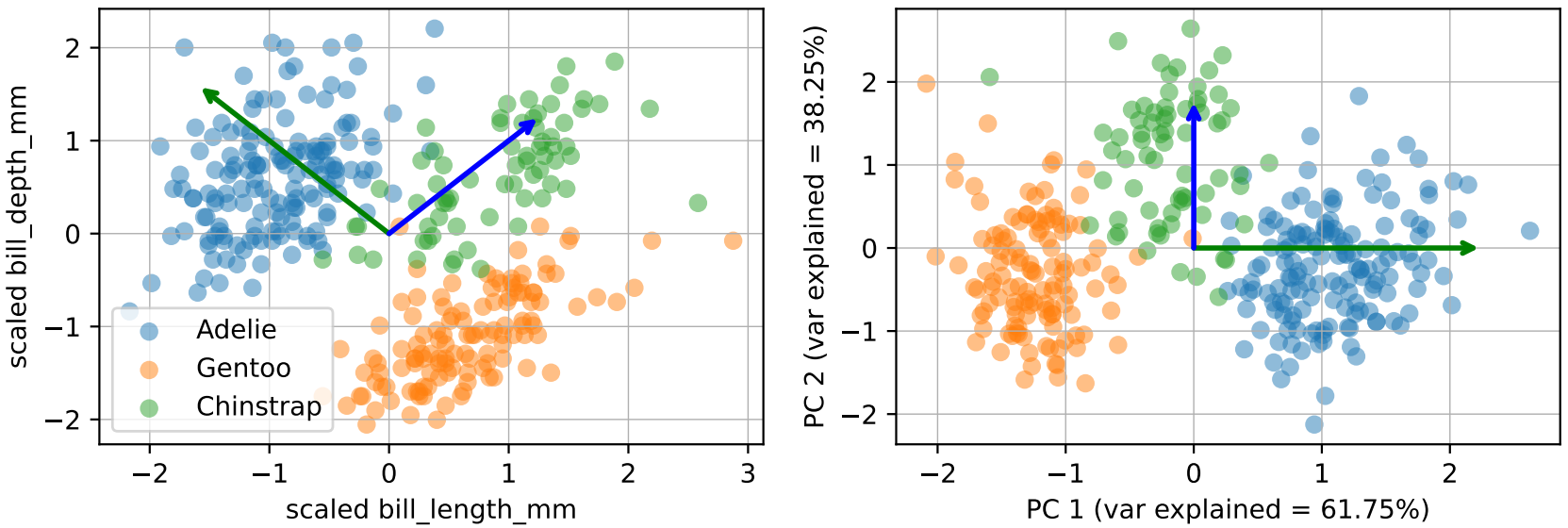


Figure: (Left) Original (but standarised) data with two choices of variables. (Right) The same but in the principal component representation.

- for this case it really is just a co-ordinate transformation

Dimension reduction: PCAs

- below shows dimension reduction (from 4 to 2)

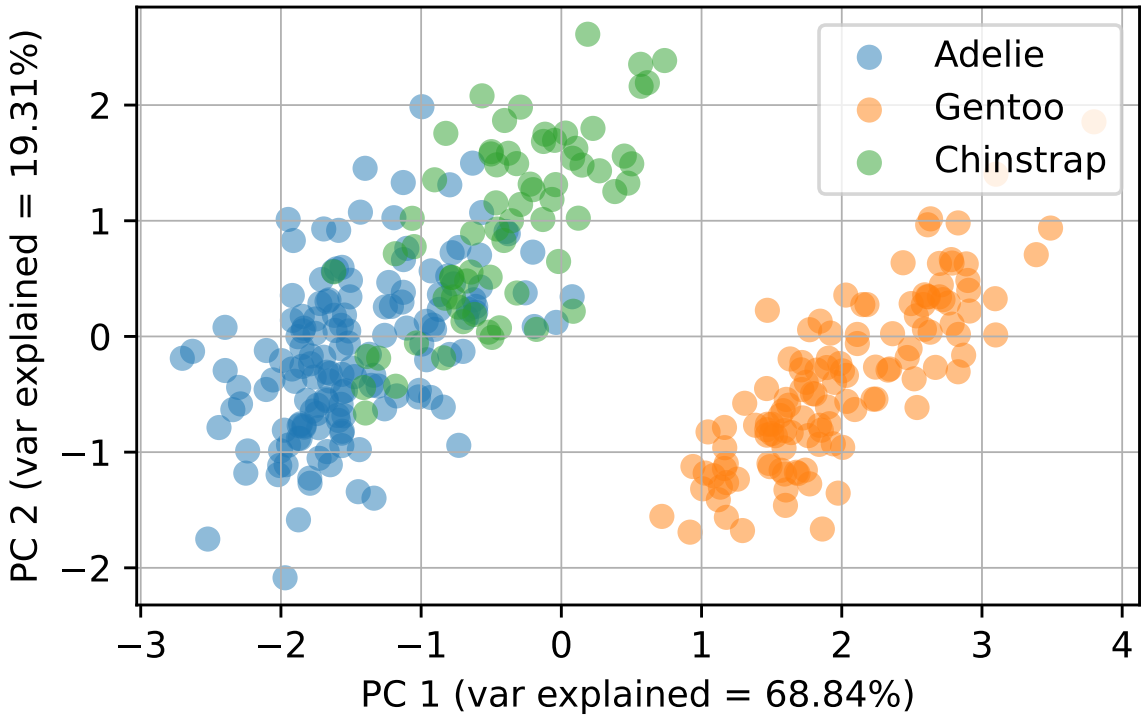


Figure: Representation in PC1 and PC2 co-ordinate system.

- note the variance explained does not sum to 100% here
  → this is now a **projection** which loses information

Dimension reduction: manifold methods

(This part makes more sense in about 15 mins)

- other manifold based methods

  → Locally Linear Embedding (preserves local distances, think local PCAs to give global co-ord)

  → *t*-distributed Stochastic Neighbor Embedding (*t*-SNE; based on pdfs, good at picking out local structures)

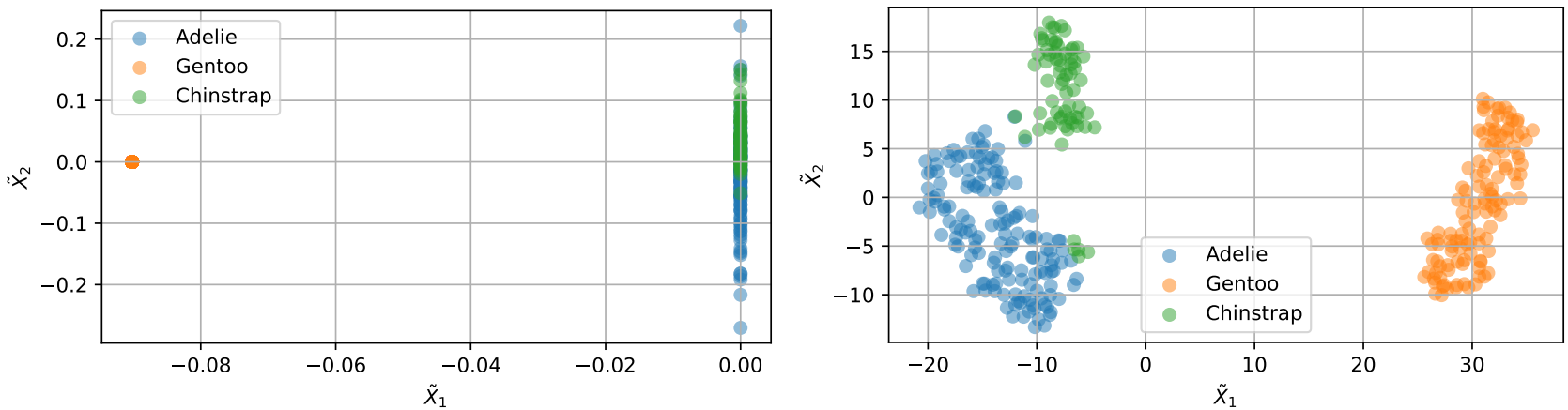


Figure: Demonstration of Locally Linear Embedding and *t*-SNE on the penguins data.

Demonstration: eigencat

- example demonstrating PCA on images of cats

  $\rightarrow$ feature extraction, cf. facial recognition
- massage data into `(n_cat,pixels)` then throw into PCA

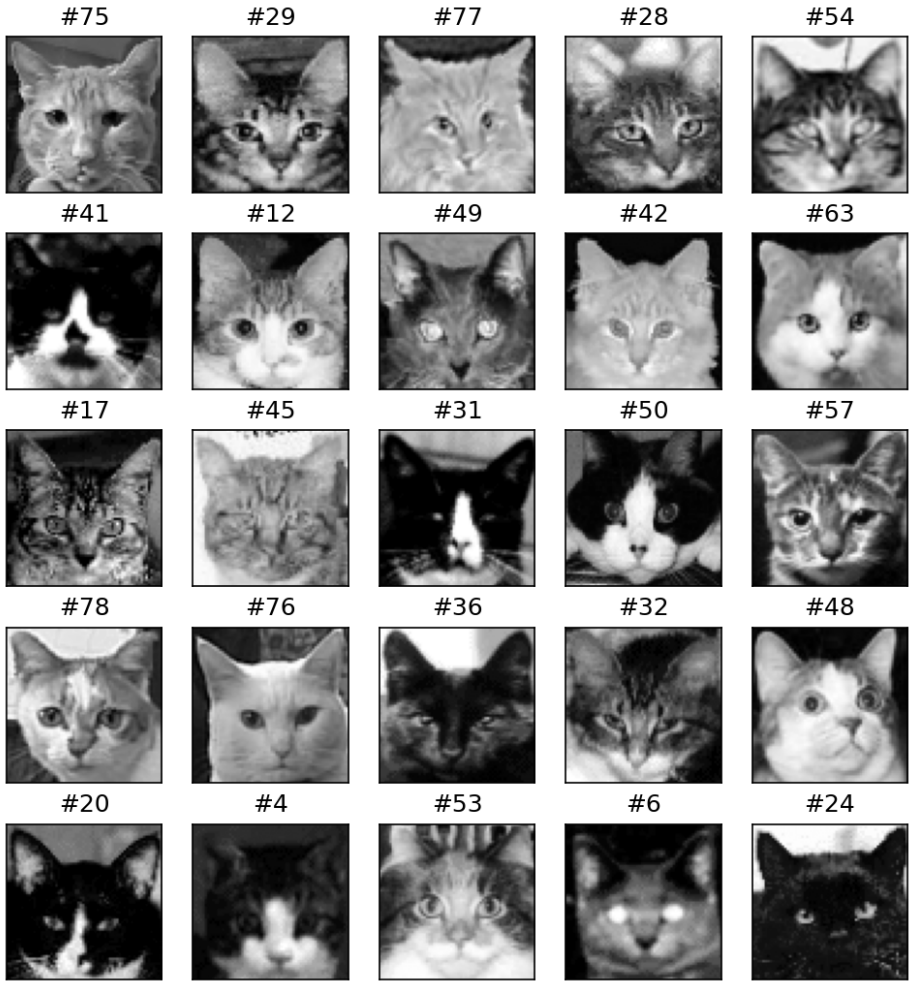


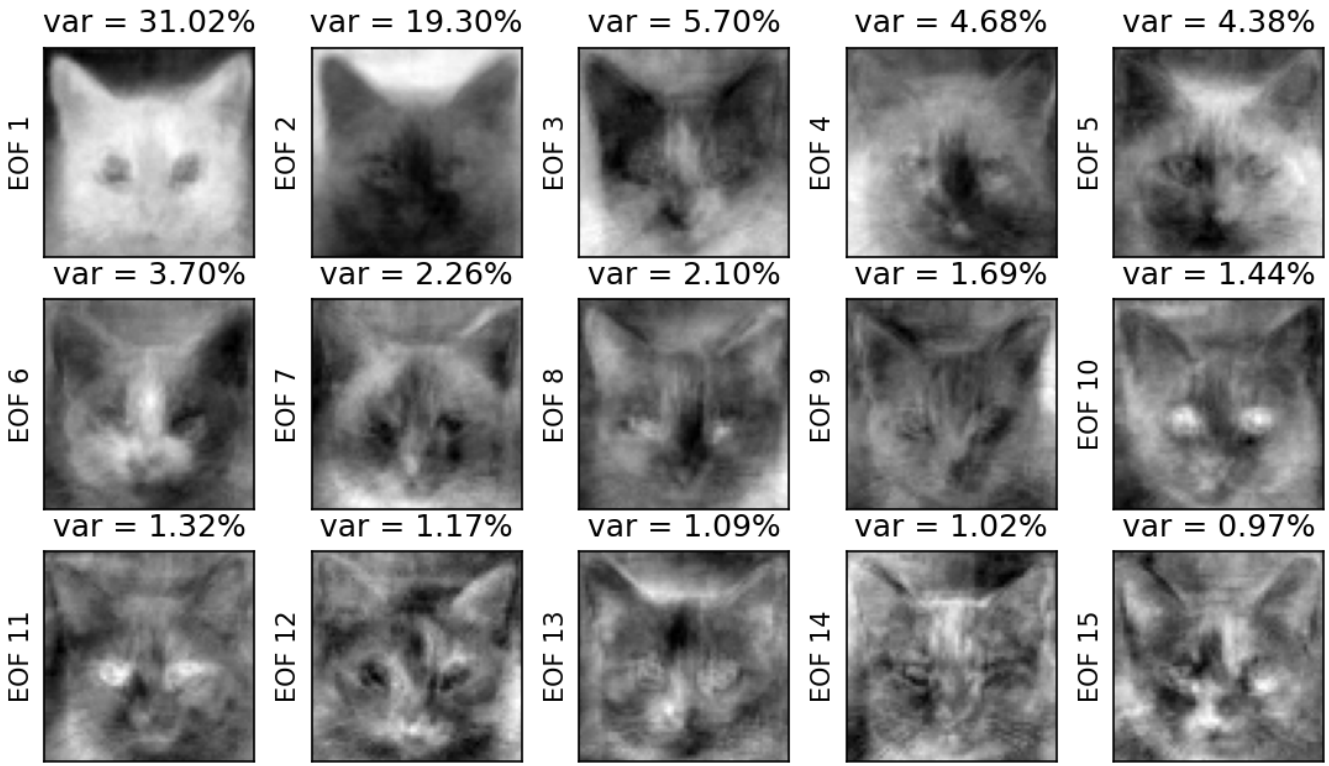
Figure: Data in `cats.csv`.

Demonstration: eigencat

- want

$$\text{Image}(\text{pixels}) = \sum_i \text{PC}_i \times \text{EOF}_i(\text{pixels}),$$

$\rightarrow$ call the patterns the EOFs, PCs are the loadings
$\rightarrow$ reshape EOFs to get "averaged" cat faces



Demonstration: eigencat

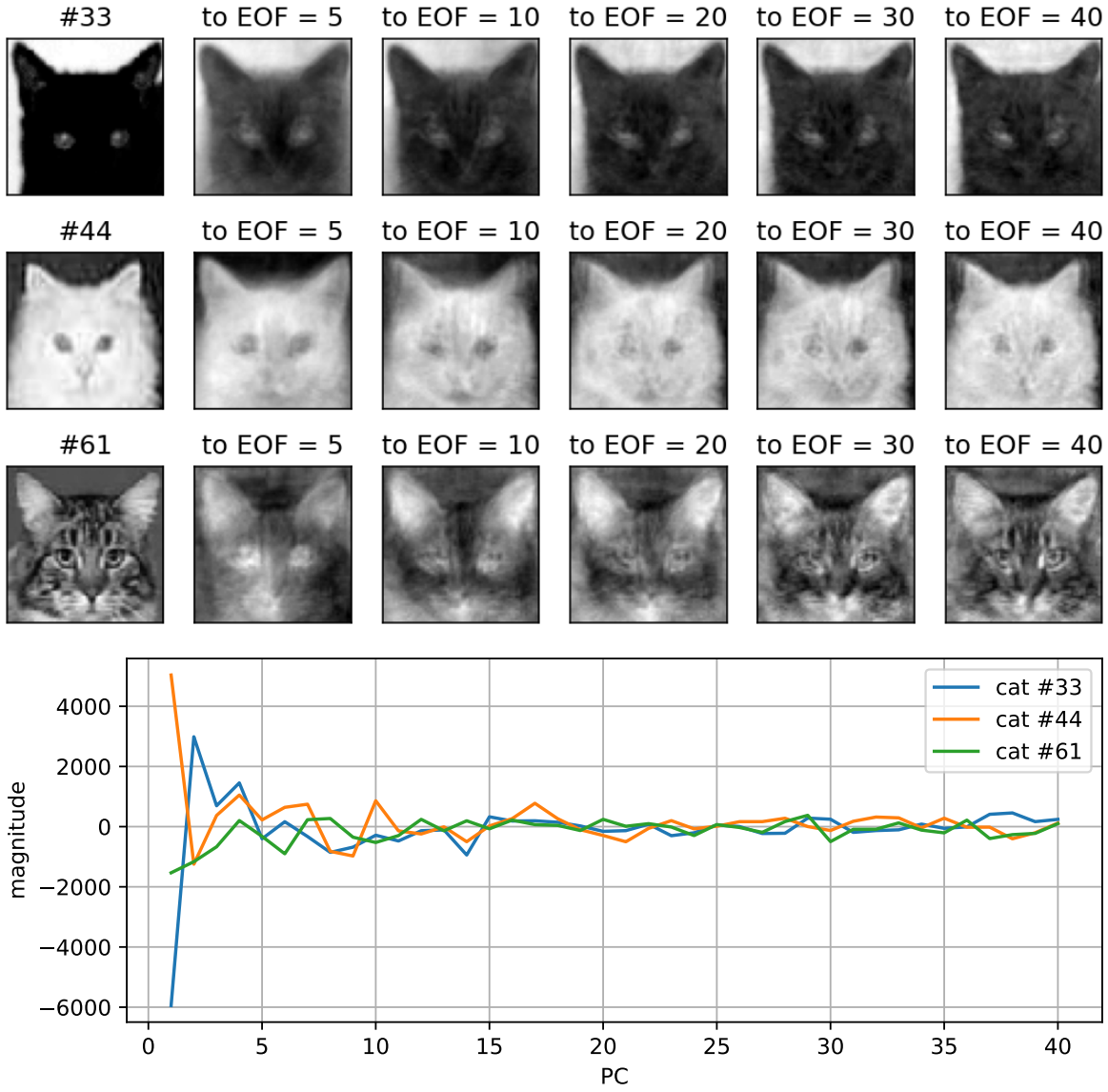


Figure: Expansion and loadings of three of the samples.

Demonstration: eigencat

- different cats will have a different PC pattern, use that as

  $\rightarrow$ signature for classification?

  $\rightarrow$ features for model training? (much lower dimension)

- warning: can start doing ridiculous things like below if my data space is large enough!

  $\rightarrow$ next few pages show the same thing if I use an expanded dataset (2000 images)

Demonstration: eigencat

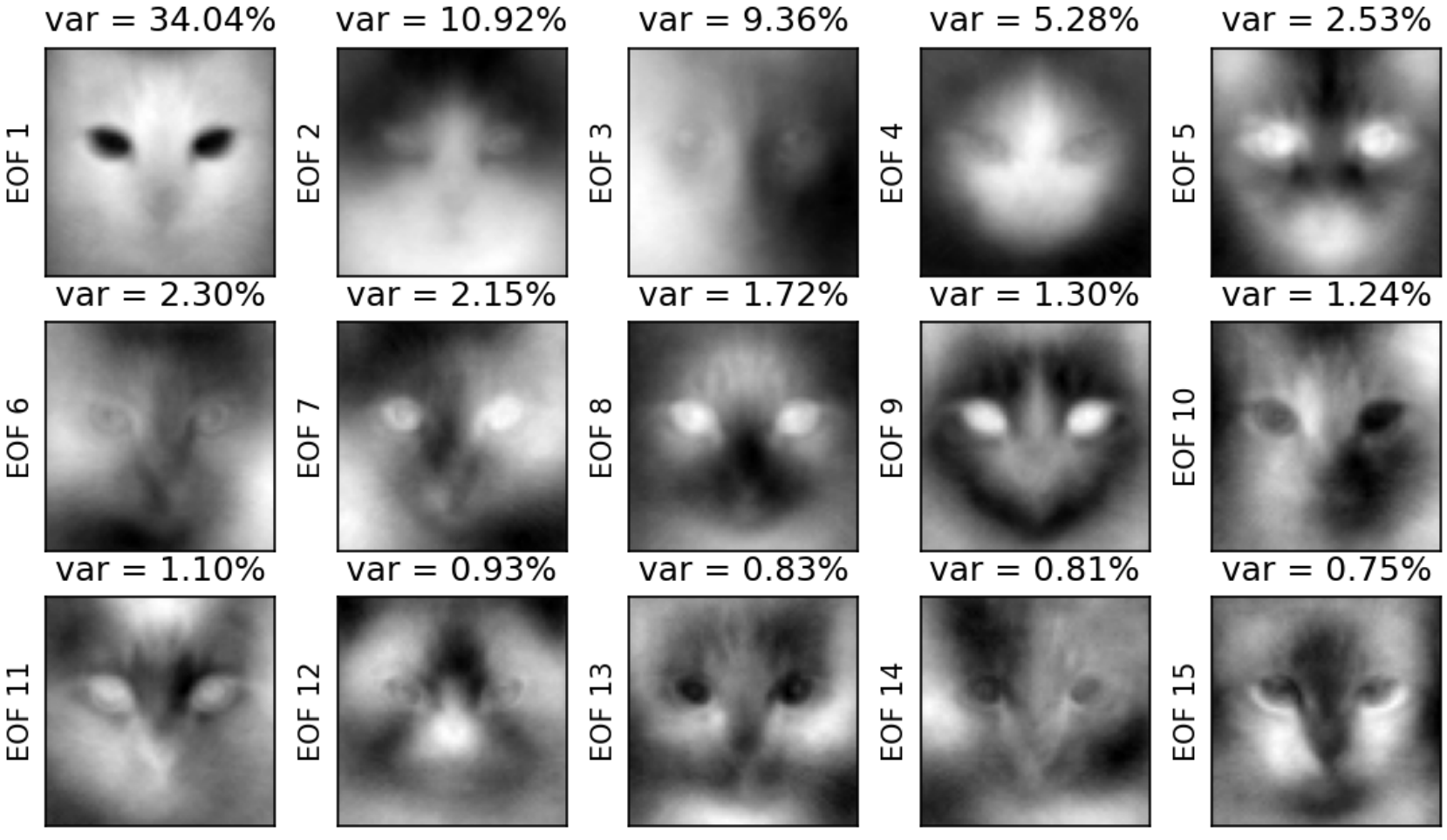


Figure: Eigencats from `cats_bw_enlarged.csv`.

Demonstration: eigencat

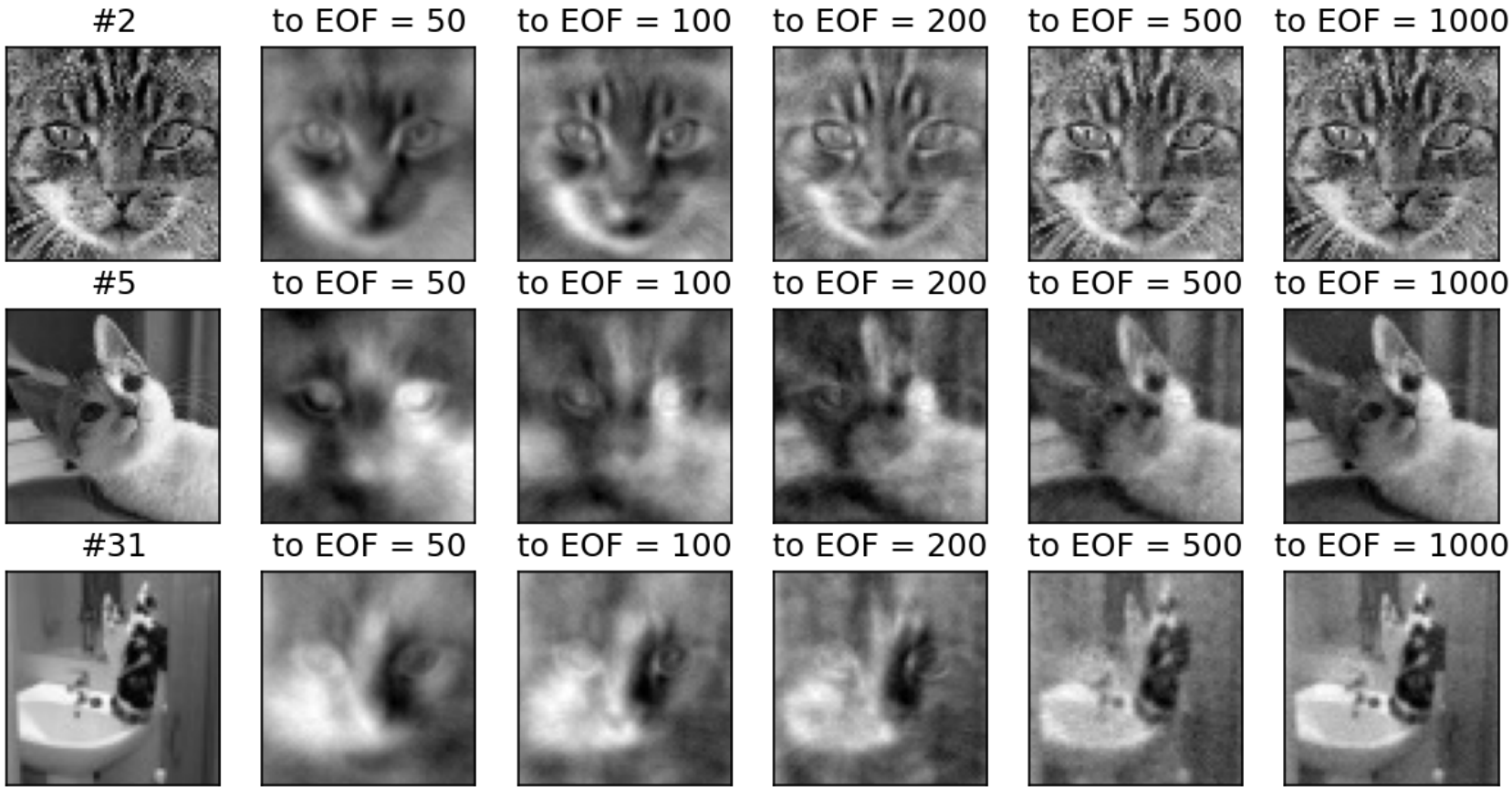


Figure: Expansion of a few selected images from the expanded dataset.

Demonstration: eigencat

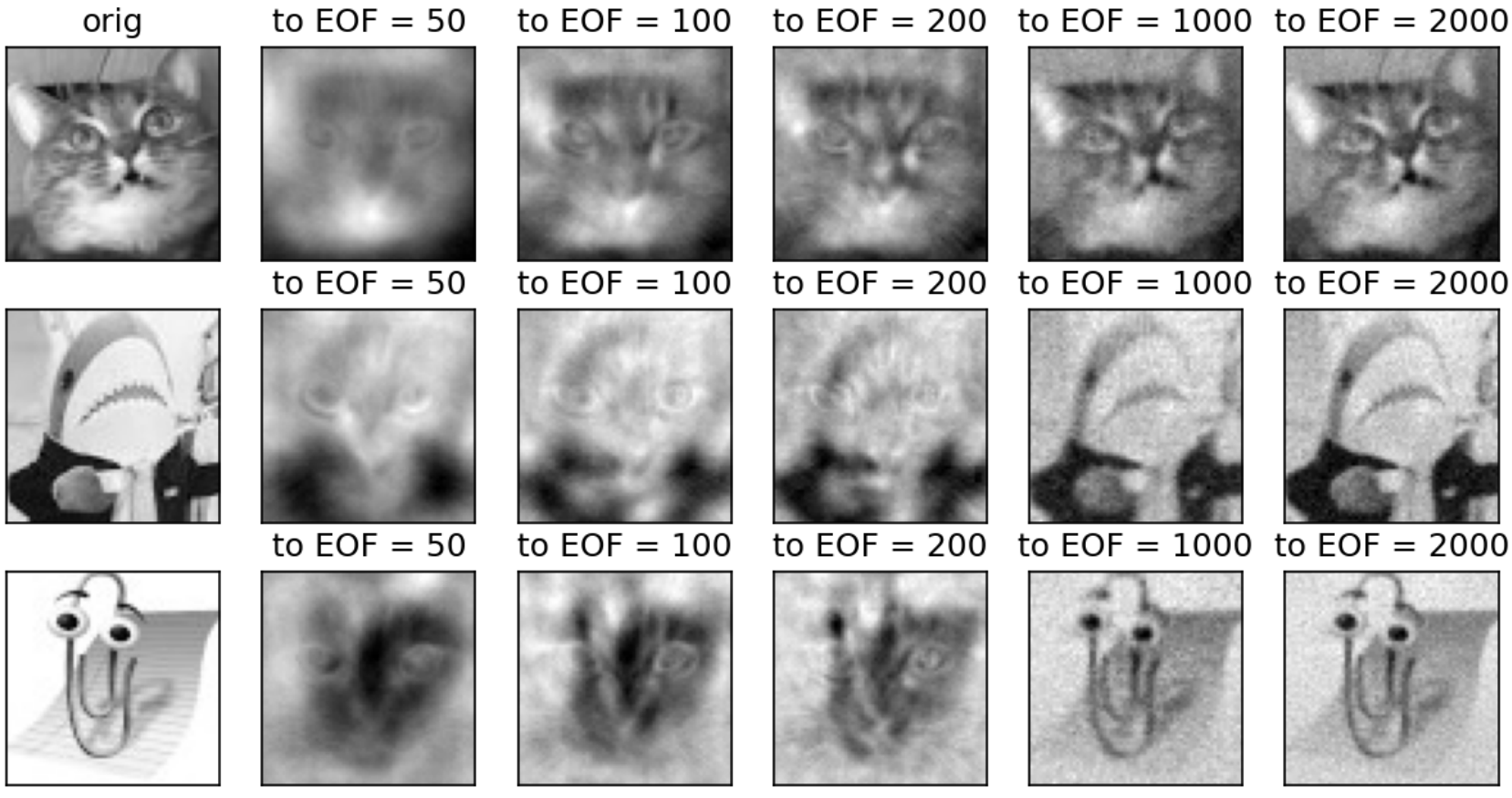


Figure: Eigencat expansion of the three ad hoc TAs (which were not in the dataset).

Clustering: a motivation

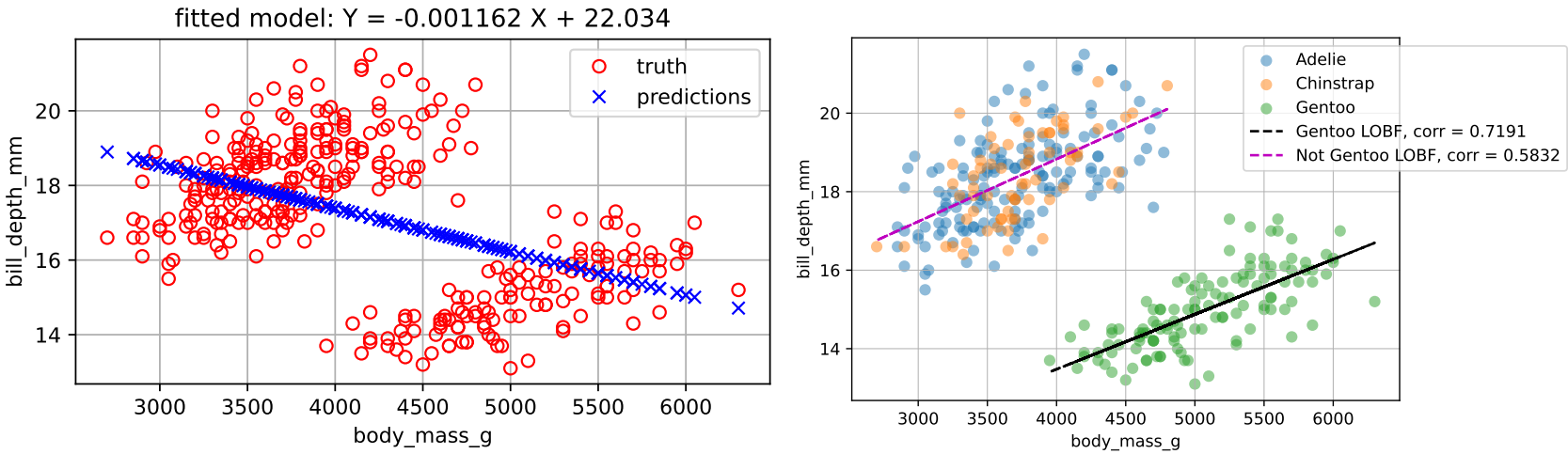


Figure: Contrived example of completely different models depending on data selection.

- if we had no labels in penguins data the linear regression is pretty bad (although it was never going to be good...)

  $\rightarrow$ use clustering to create labels?

  $\rightarrow$ e.g. use that to inform training of model, or to train different models

Oceanographic examples

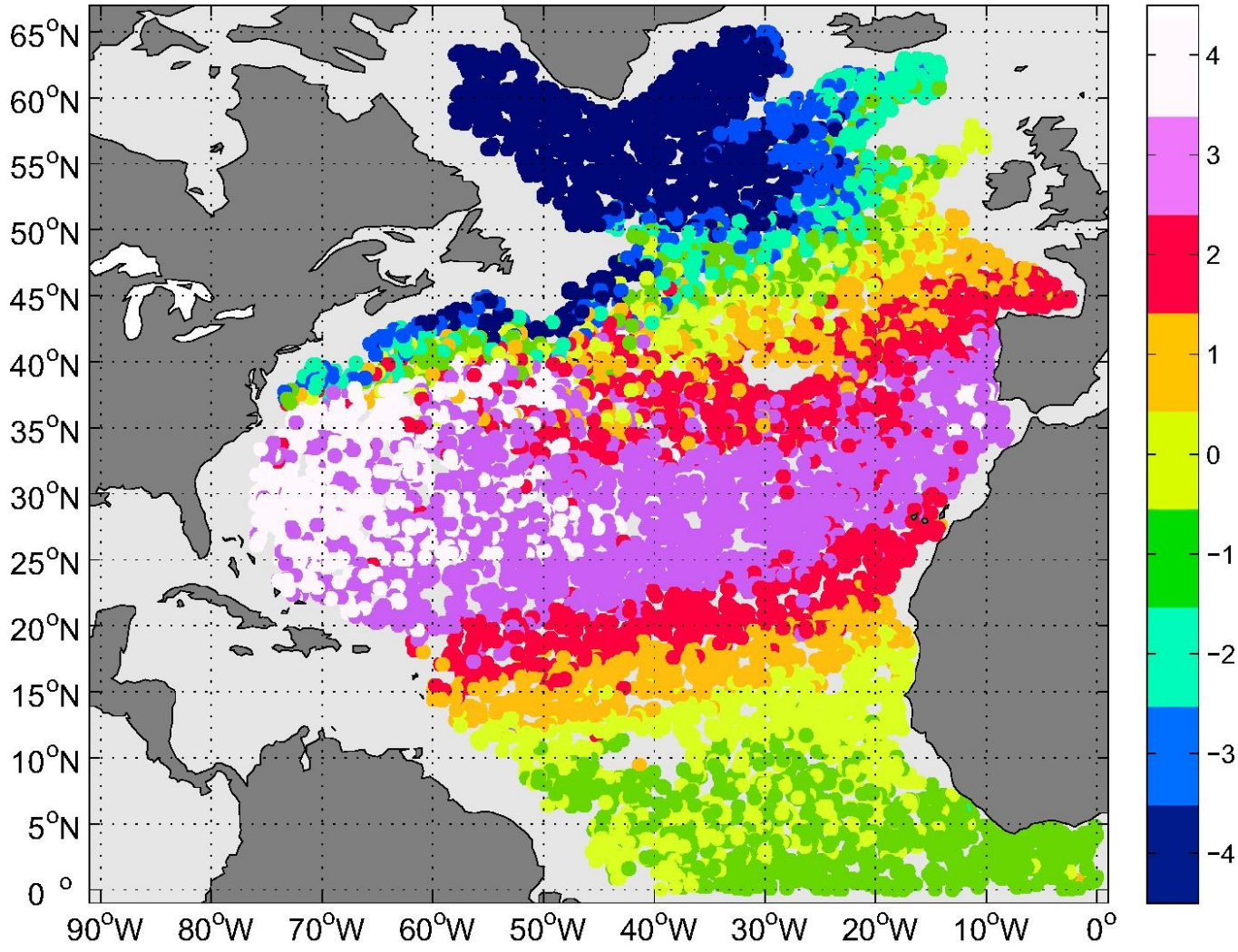


Figure: From Maze *et al.* (2017), Fig. 4. Gaussian Mixture Model to identify watermass clusters from Argo data in Atlantic.

Oceanographic examples

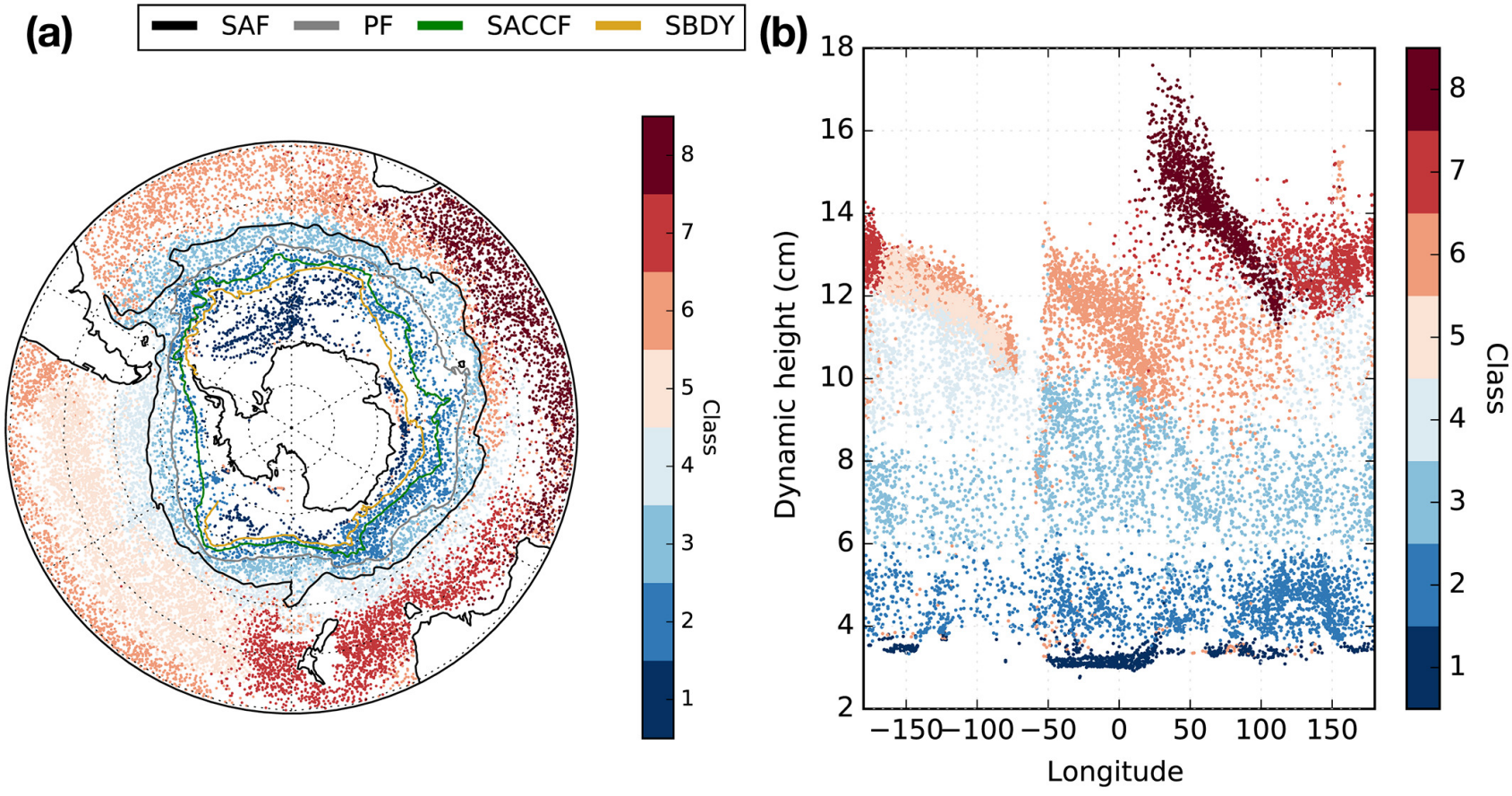


Figure: From Jones *et al.* (2019), Fig. 5. Gaussian Mixture Model to identify watermass clusters from Argo data in Southern Ocean.

Oceanographic examples

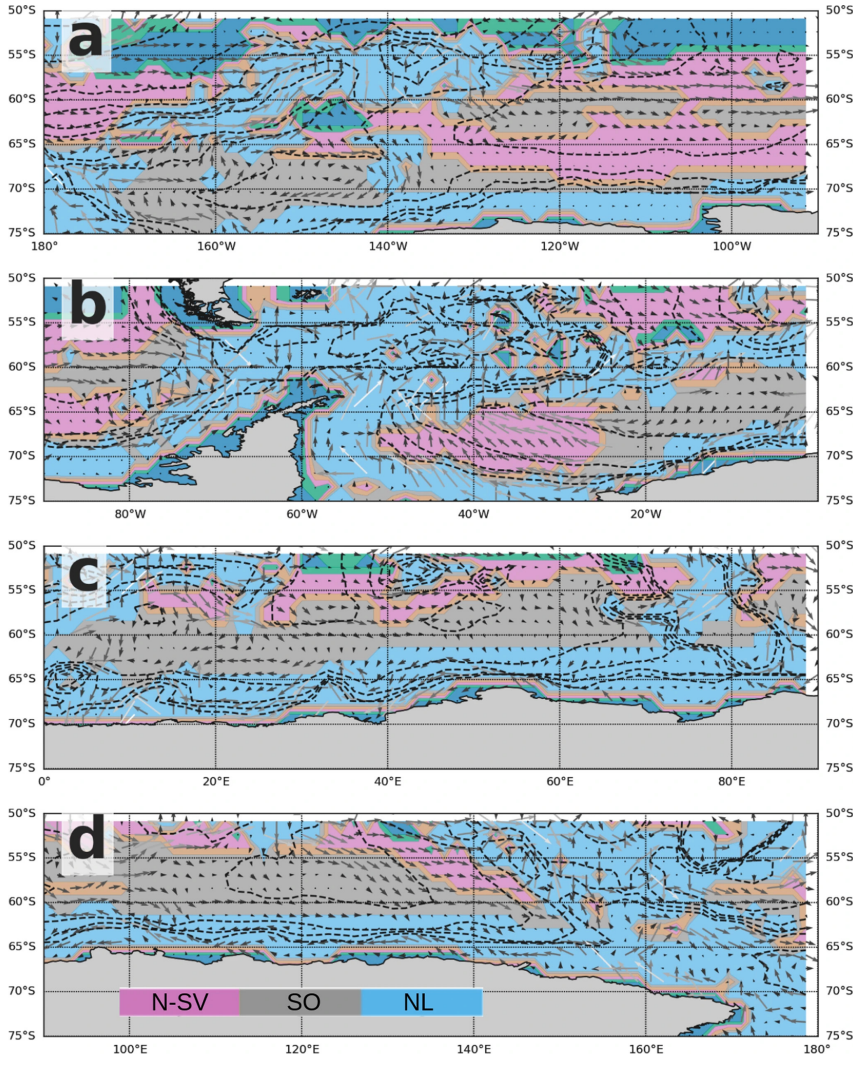


Figure: From Sonnewald *et al.* (2023), Fig. 4. *k*-means to identify clusters based on dynamic (from barotropic vorticity budget).

Oceanographic examples

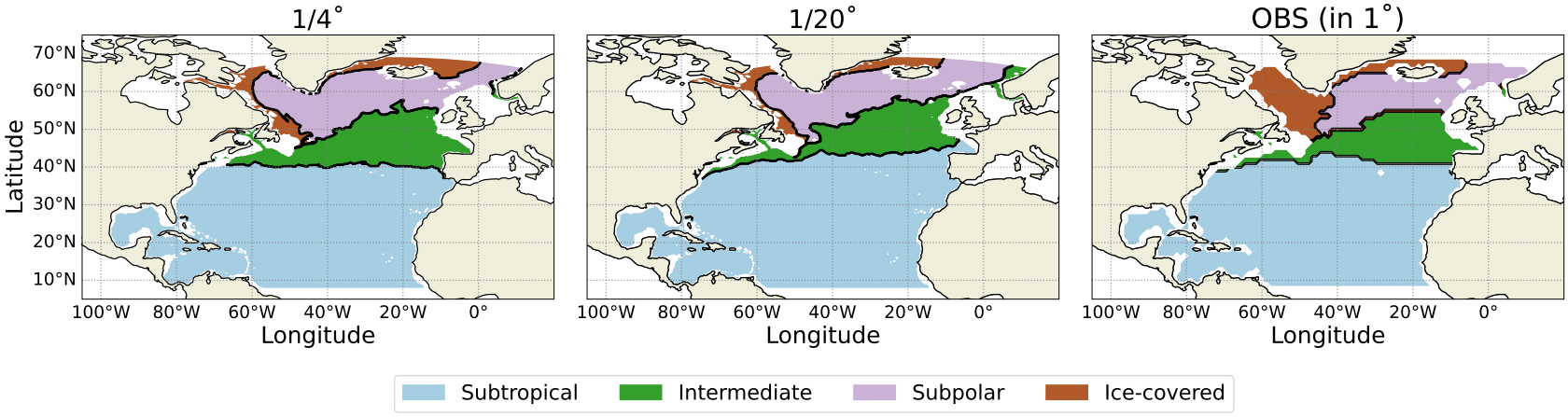


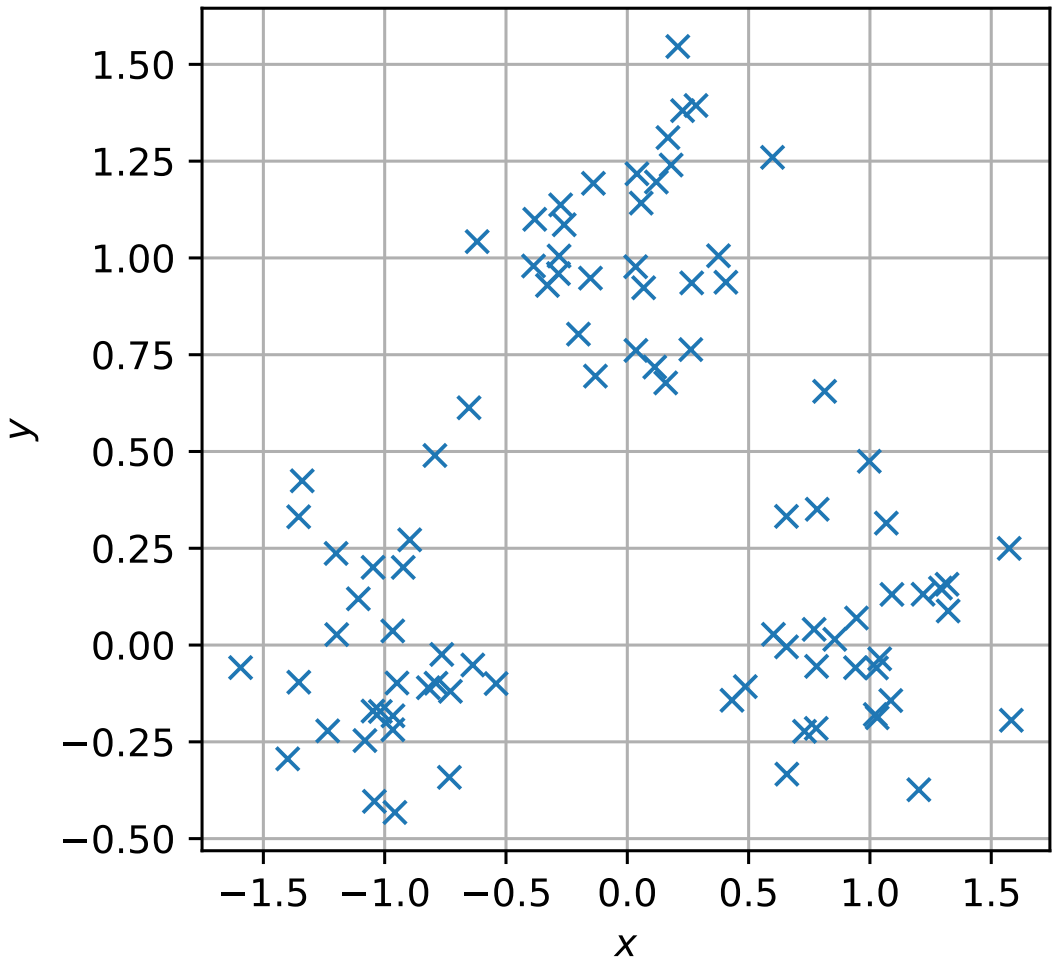
Figure: From Ruan *et al.* (in prep.), which identifies regions depending on biogeochemical activity.

- hierachy building of clusters using various variables
  → SST, MLD, sea-ice fraction, not (lon, lat)
  → nitrate concentation, surface chlorophyll concentration

- aim to classify POC and DOC export, but do not include those in building of cluster

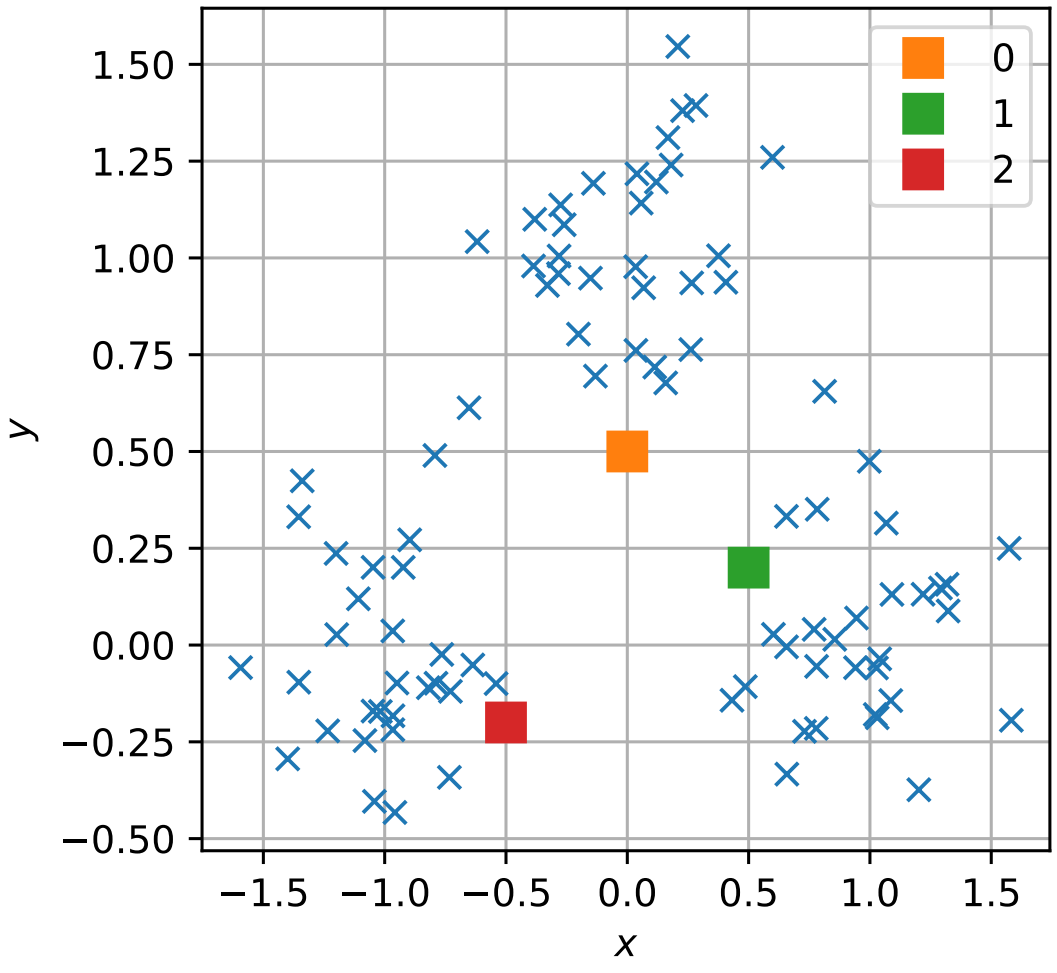
e.g. K-means

- demonstrate the algorithm with K-means
  $\rightarrow$ artificially create some data ($K = 3$ is sensible)



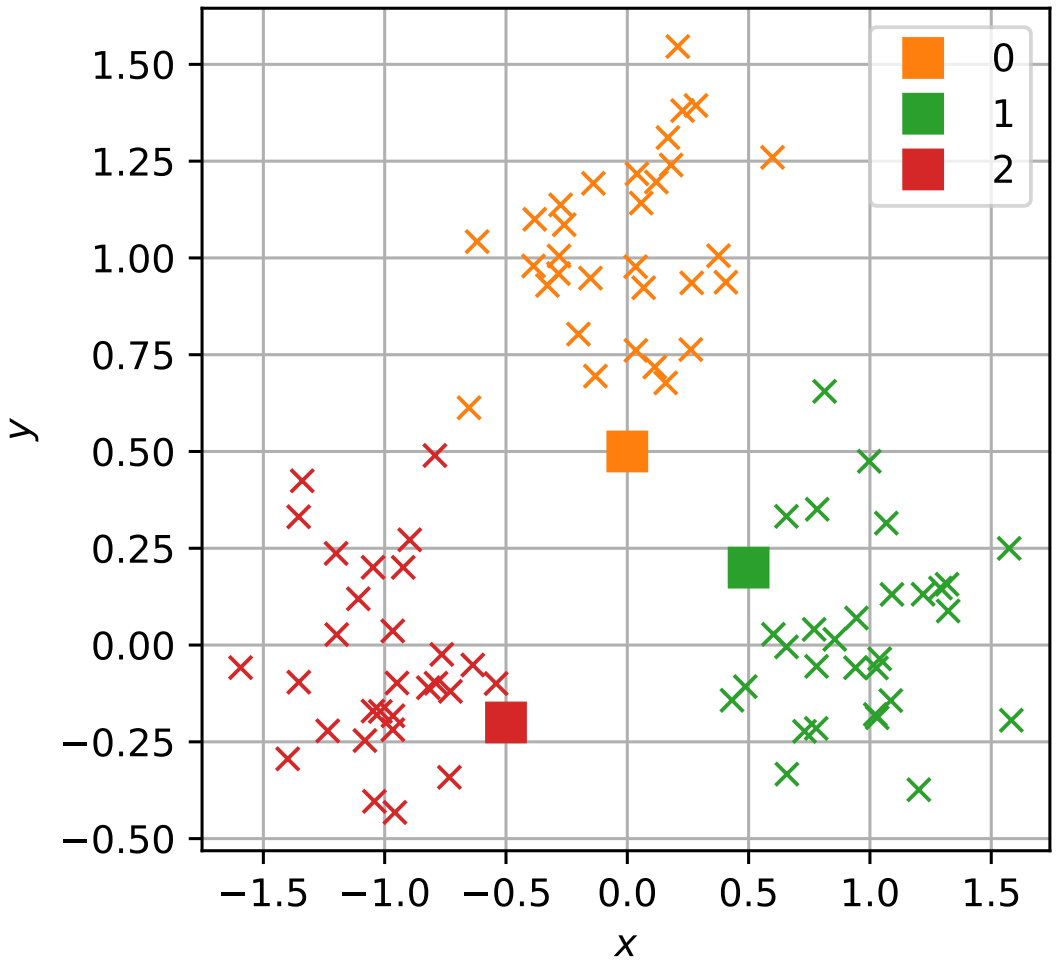
e.g. K-means

- provide initial centres of clusters
  $\rightarrow$ K-means is quite robust, can just guess



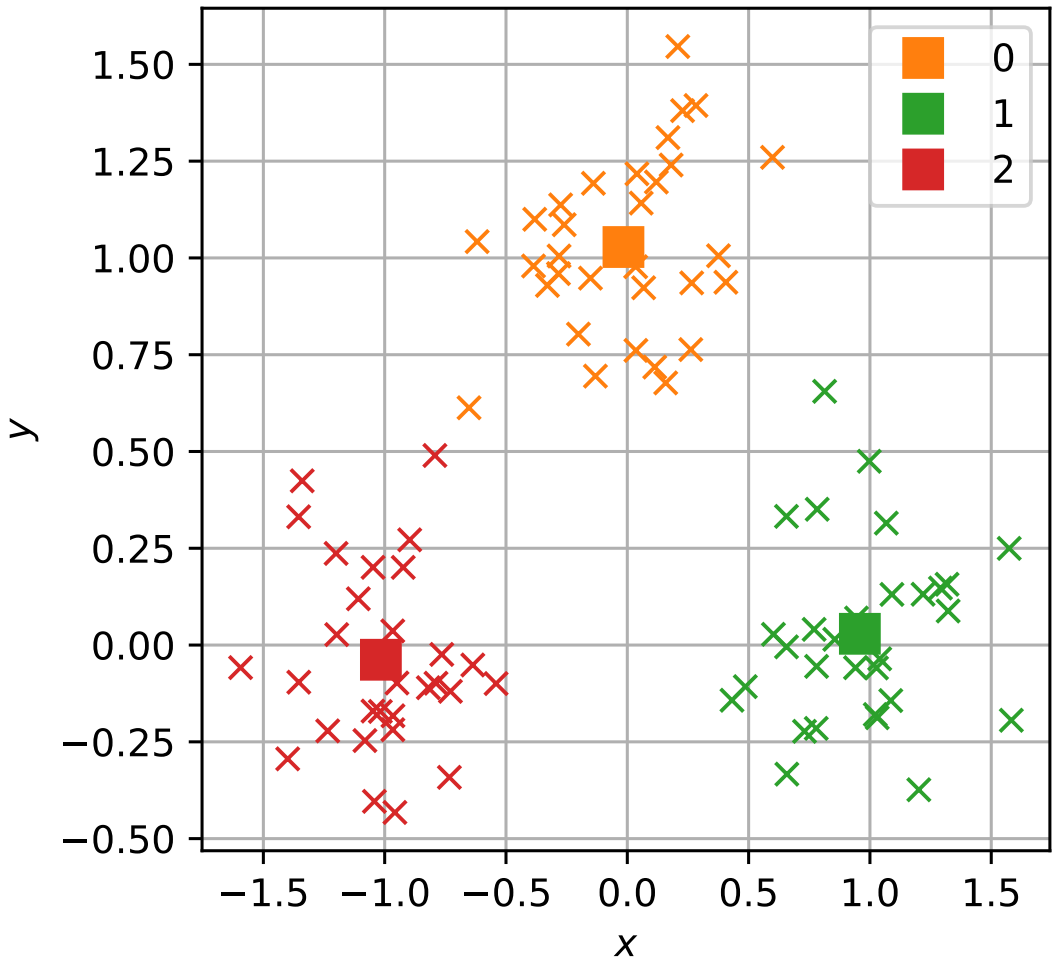
e.g. K-means

- label points closest to centres accordingly
  $\rightarrow$ distance dependent, normally L^2 (Euclidean distance)



e.g. K-means

- from new cluster, find the new location of the centre
  $\rightarrow$ update and iterate accordingly



K-means in `sklearn`

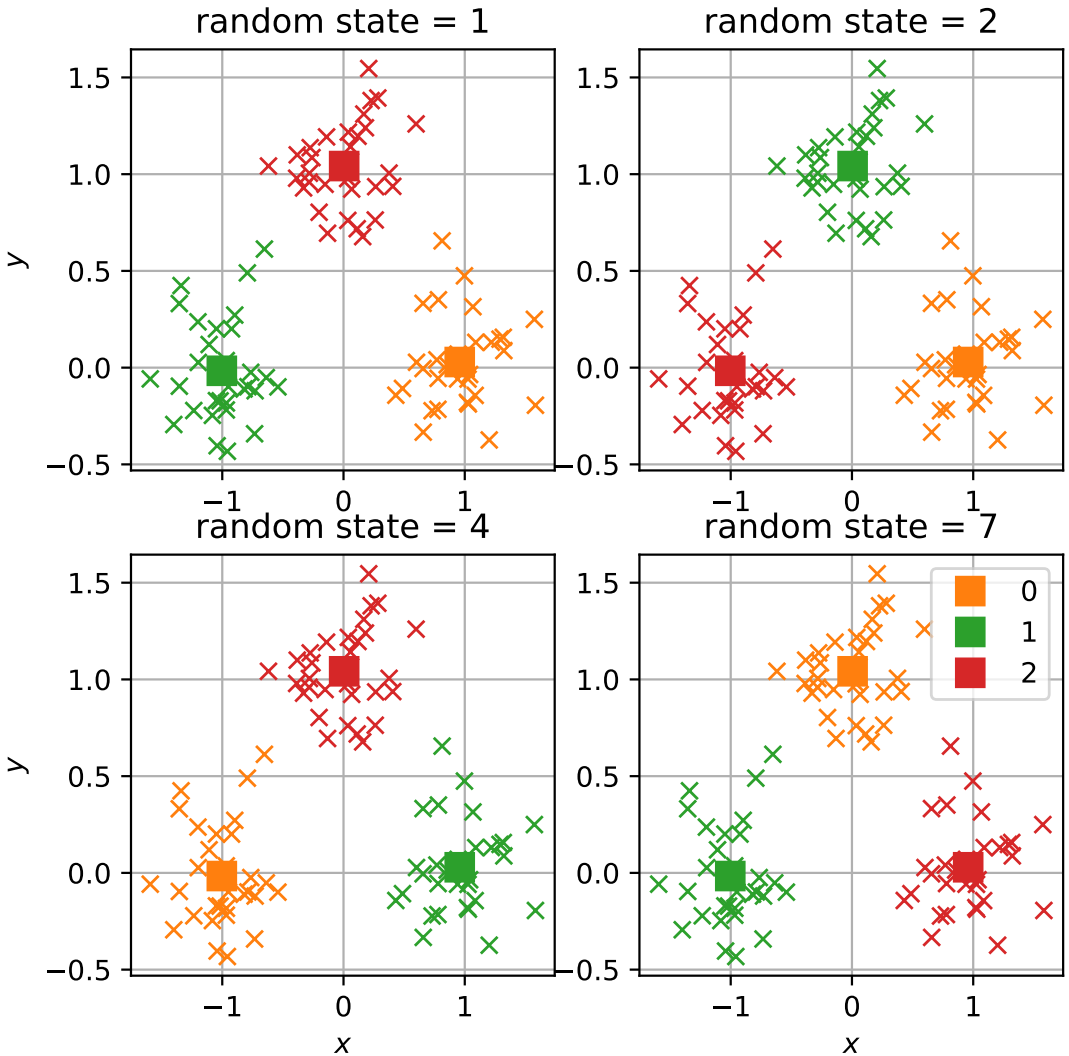


Figure: *K*-means through `sklearn`. Random state changes initial guess.

A case where K-means as is will never(?) work

- all of the above depends on the choice of 'distance' again

A case where K-means as is will never(?) work

- all of the above depends on the choice of 'distance' again
- there are well-known examples where the L^2 distance is simply not the relevant one
  $\rightarrow$ e.g. the moon data (2d example) and Swiss roll data (can do 2d or 3d)

Figure: Moon Moon and two rolled up towels resembling a Swiss roll.

A case where K-means as is will never(?) work

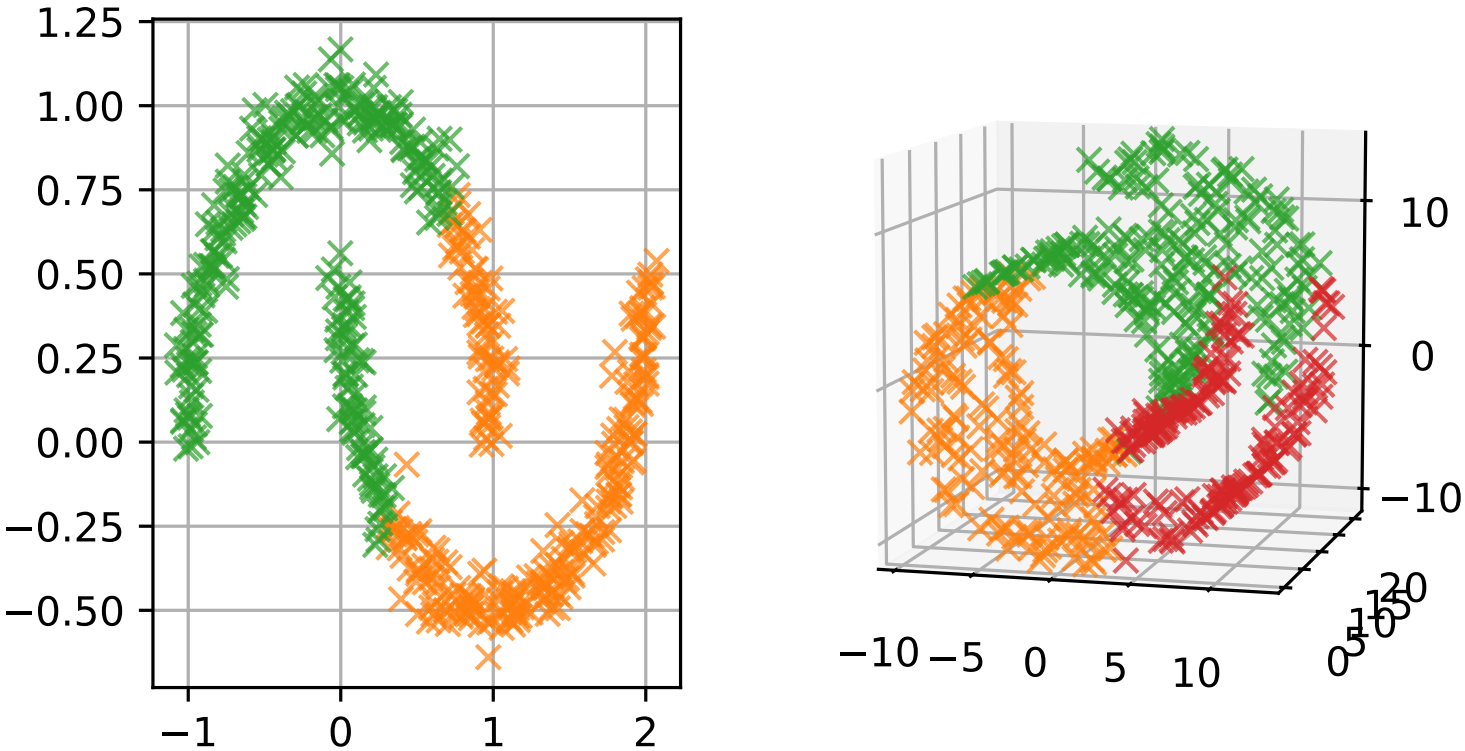


Figure: K-mean as applied to the crescent moons and swiss roll data.

The issue of 'distance'

- the data may have structure and lives on a curve/surface/volume etc. (e.g. manifold up to noise)

  $\rightarrow$ manifold may have a lower dimension structure (1d and 2d here)

  $\rightarrow$ but embedded in an ambient space ($\mathbb{R}^2$ and $\mathbb{R}^3$ here)

The issue of 'distance'

- the data may have structure and lives on a curve/surface/volume etc. (e.g. manifold up to noise)

  $\rightarrow$ manifold may have a lower dimension structure (1d and 2d here)

  $\rightarrow$ but embedded in an ambient space ($\mathbb{R}^2$ and $\mathbb{R}^3$ here)
- it is the distance intrinsic to the manifold that is presumably of interest

  $\rightarrow$ L^2 is the distance extrinsic to manifold and inherited from the ambient/embedding

- find a suitable embedding, as in dimension reduction previously?

  $\rightarrow$ reduce dimension first, then cluster

Moon data demonstration

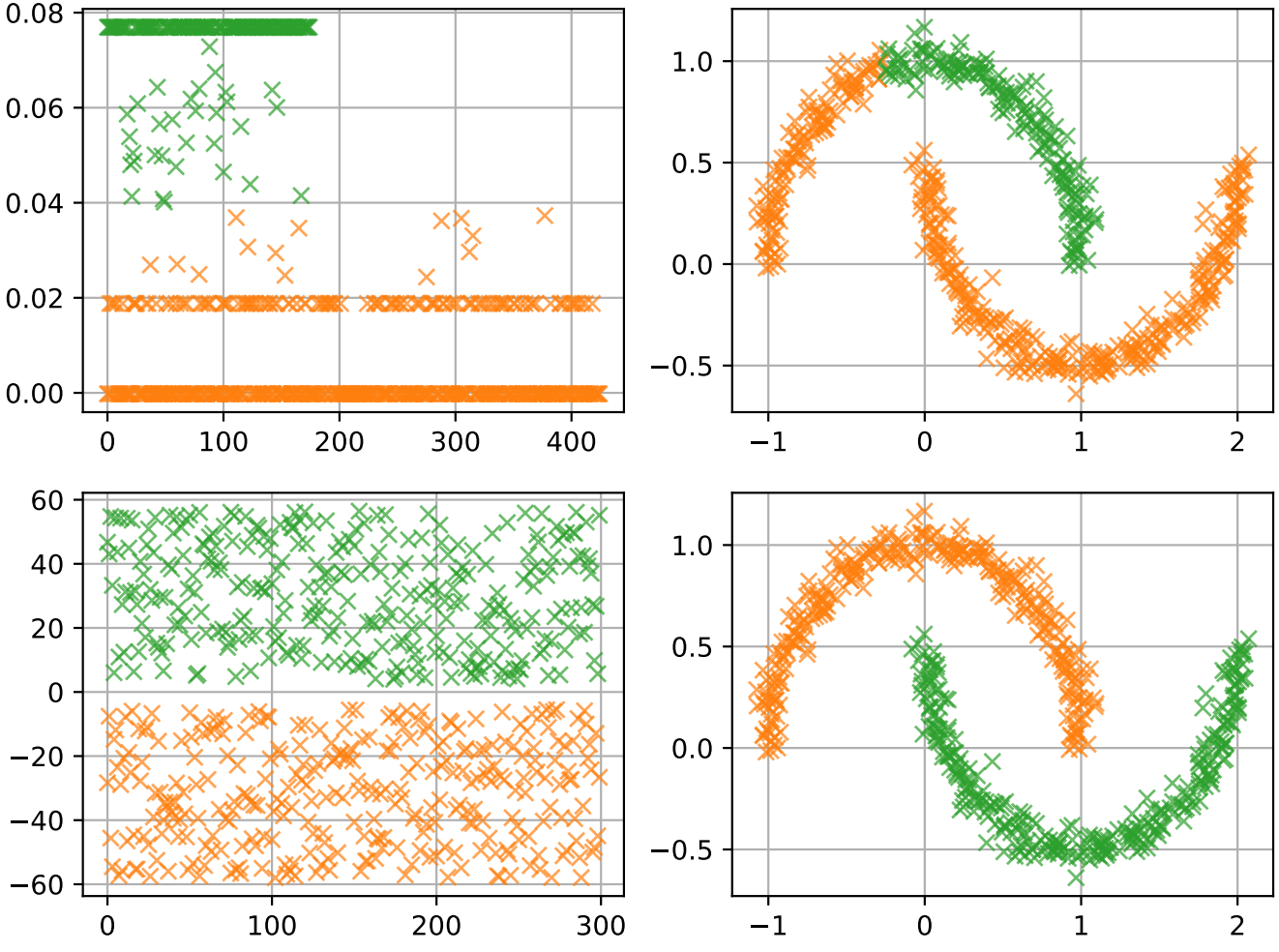


Figure: 1d projection (of 2d data) via (top) LLE and (bottom) *t*-SNE before *K*-means. LLE can work on occasions, but *t*-SNE seems more robust.

Swiss roll data demonstration

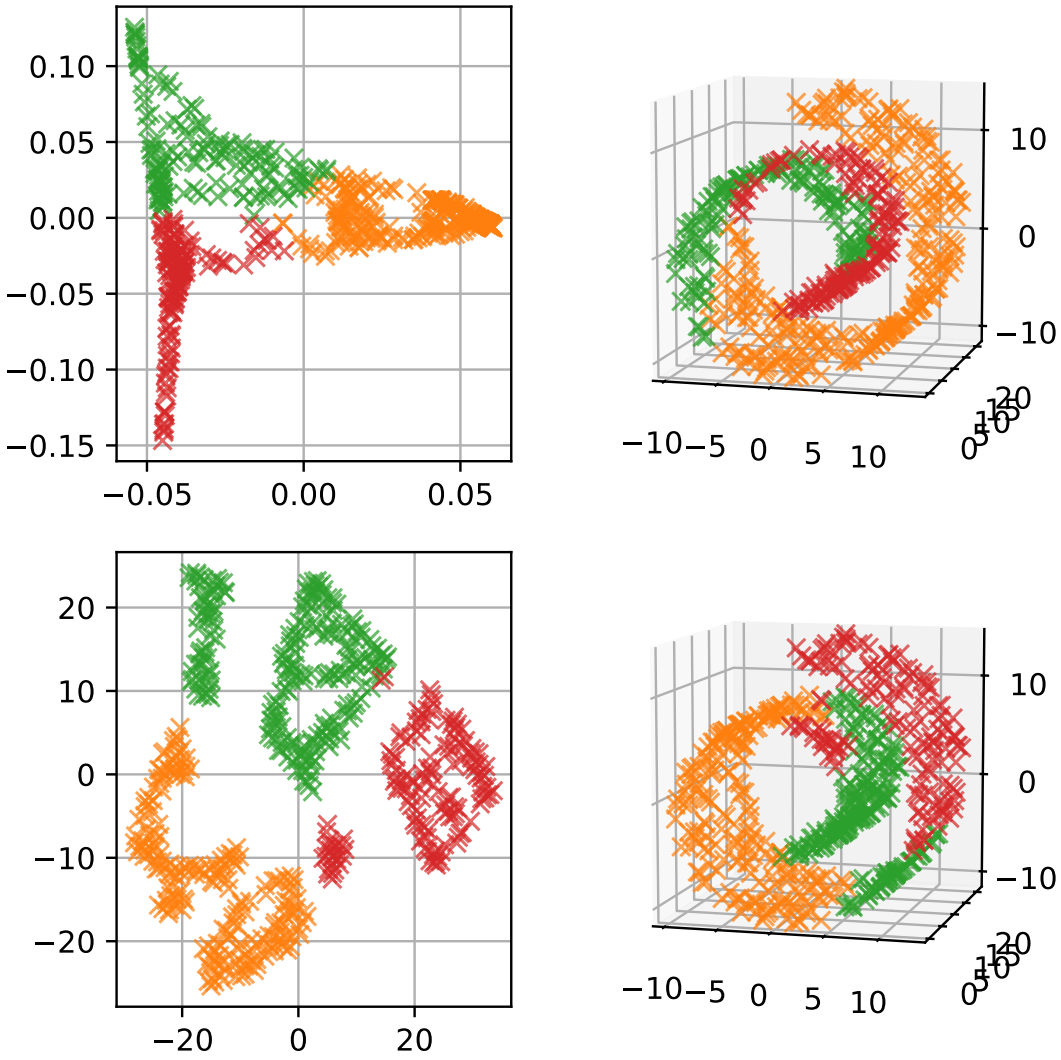


Figure: 2d projection (of 3d data) via (top) LLE and (bottom) t-SNE before K-means. LLE arguably segments better.

Other algorithms (e.g. DBSCAN)

- other algorithms do things differently
- DBSCAN considers clusters as high density regions and gaps are low density regions

  $\rightarrow$ two parameters: radius ϵ of some ball and number of points within said ball to quantify 'density'

  $\rightarrow$ number of clusters identified is a result of the above two choices (unlike in K-means)

  $\rightarrow$ can deal with non-Euclidean distances (you need to provide it though)

- hyper-parameter tuning + cross-validation needed!

Other algorithms (e.g. DBSCAN)

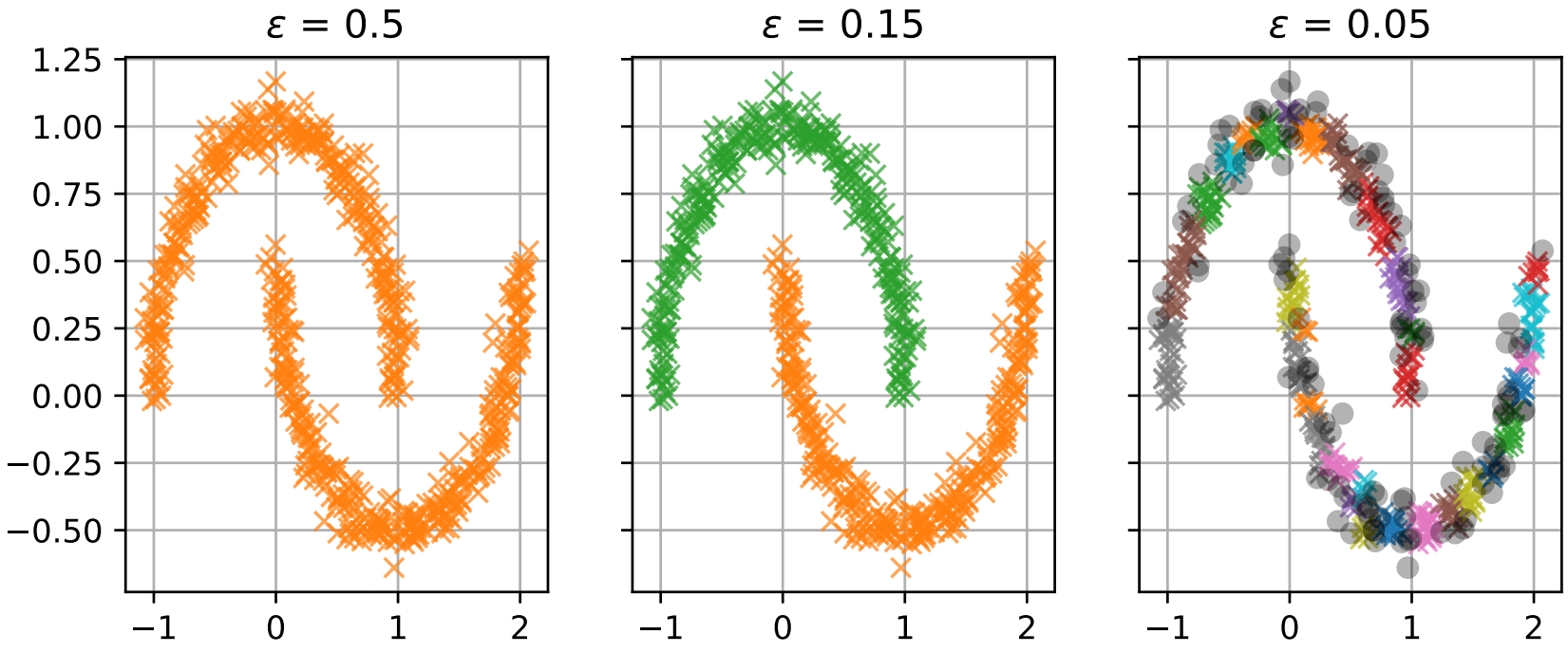


Figure: DBSCAN on moon data at varying ϵ.

- optimal ϵ for the two clusters
- if too small the too many clusters identified

  $\rightarrow$ black points are 'noise' points (no confidence in which cluster it should fall in)

Other algorithms (e.g. DBSCAN)

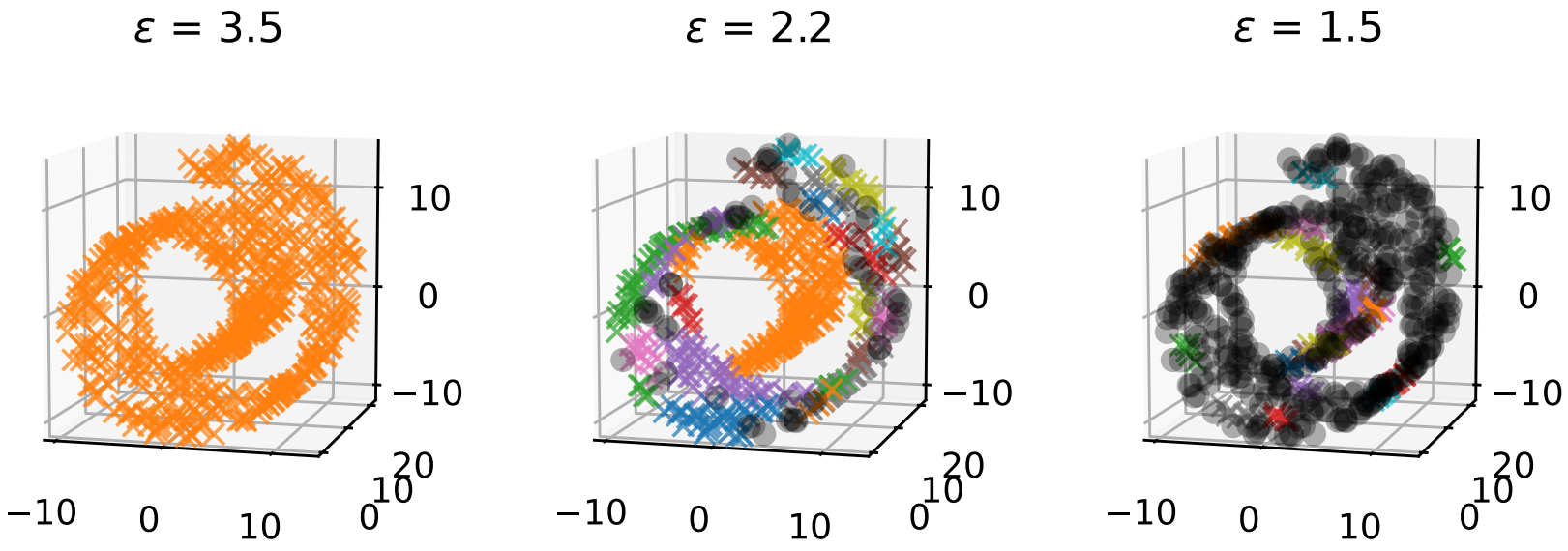


Figure: DBSCAN on swiss roll data at varying ϵ.

- optimal ϵ for the one giant cluster (bit contrived though...)
- smaller ϵ leads to segmenting along the surface, so ok

Demonstration: penguins data

- K-means ($K = 3$) on full 4d data (standardised per feature), then compare classification skill
  $\rightarrow$ some manual remapping needed
  $\rightarrow$ skill is around 91% accuracy for this realisation

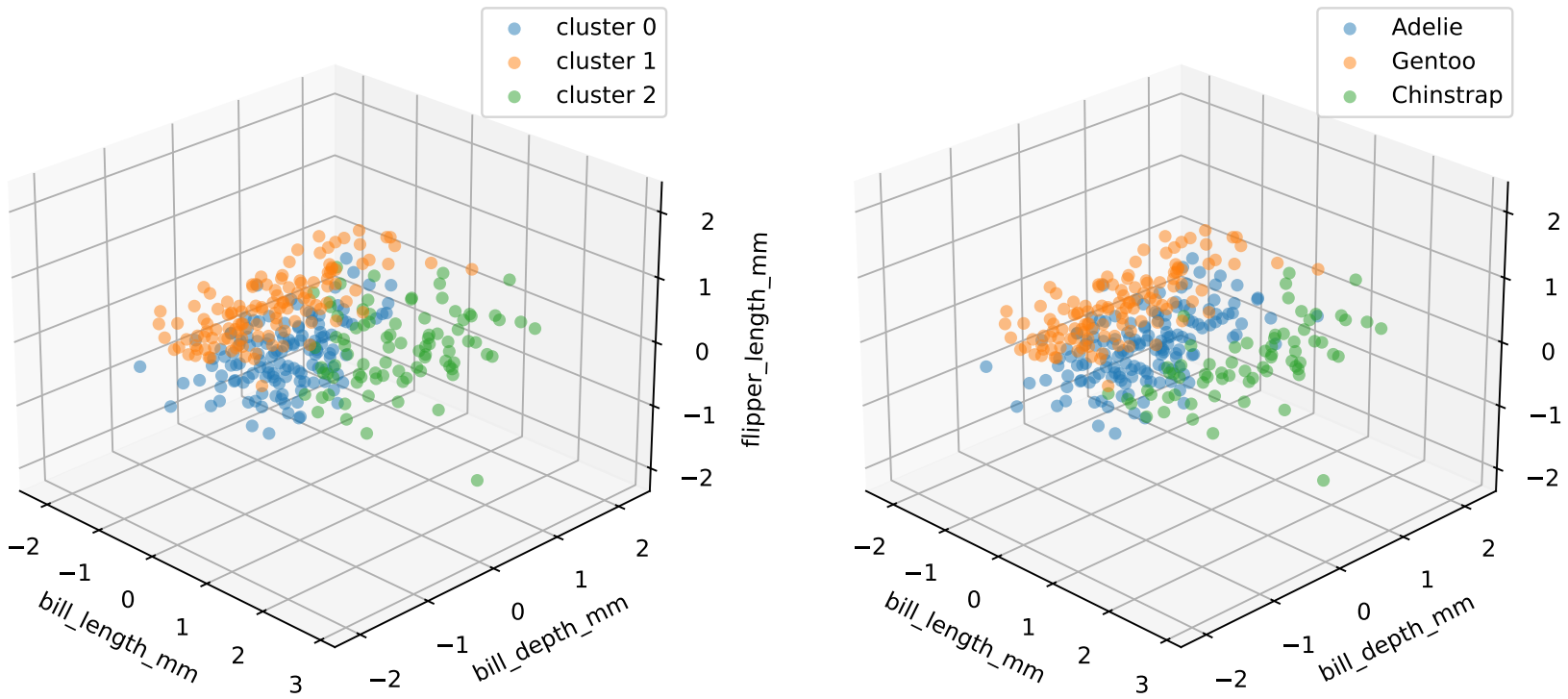


Figure: K-means on standardised penguins data.

Demonstration: penguins data

- as above but with a t-SNE to 2d before K-means

  $\rightarrow$ some manual remapping needed

  $\rightarrow$ skill is around 96% accuracy for this realisation

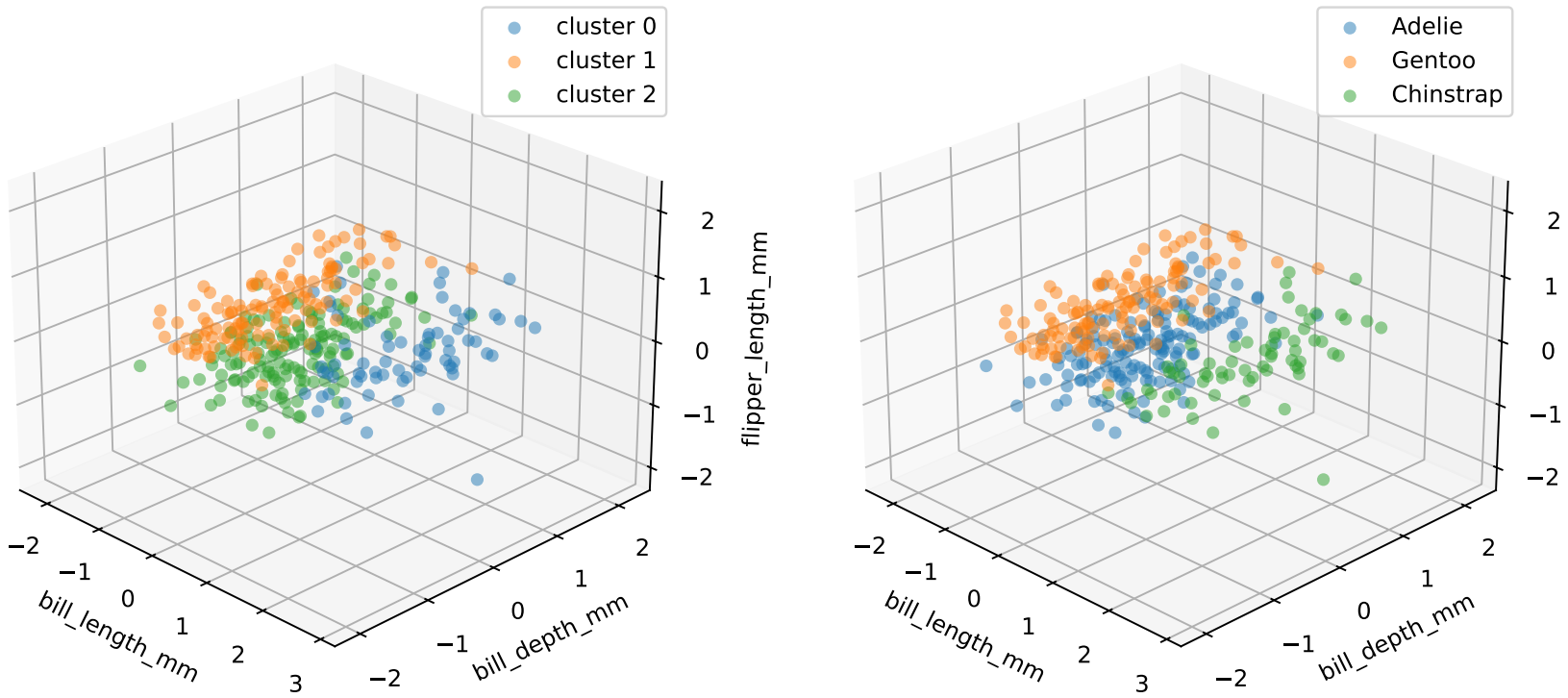


Figure: t-SNE to 2d before K-means on standardised penguins data.

Validation and model robustness

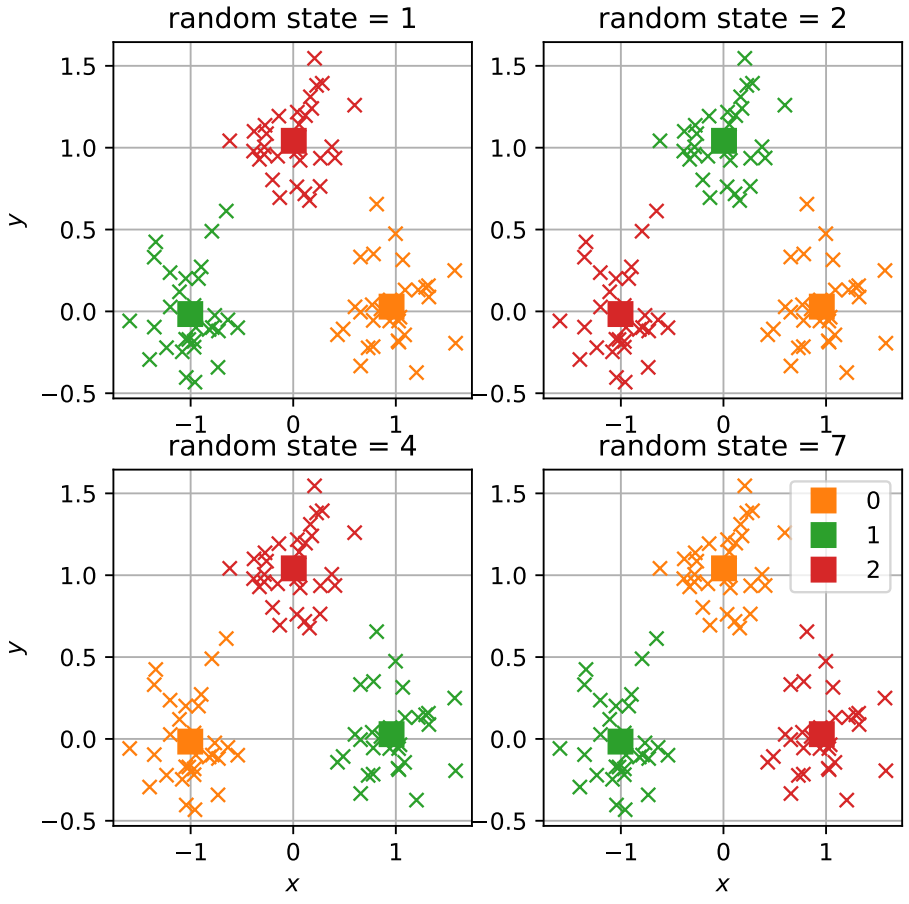


Figure: *K*-means through `sklearn`. Random state changes initial guess.

- there is inherent randomness in the training

  → e.g. from train-test split

  → resulting models can have some randomness also

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

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 do with machine learning

Cross-validation

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

  $\rightarrow$ particularly if testing data is outside the pdf of the training data
- e.g. high degree polynomial fitting

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

  $\rightarrow$ the degree of polynomial p could be regarded as the model hyperparameter here

  $\rightarrow$ 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

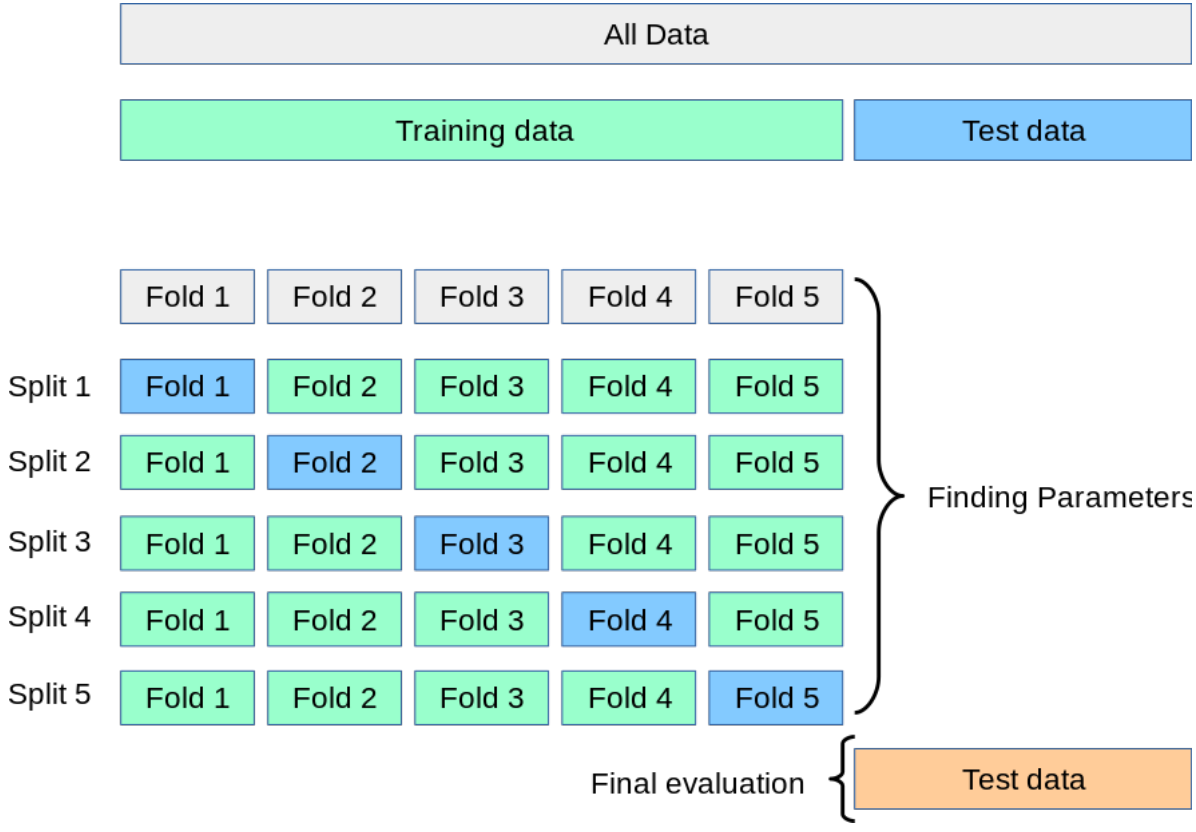


Figure: Schematic of k-fold cross-validation. Taken from `https://scikit-learn.org/stable/modules/cross_validation.html`.

- 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

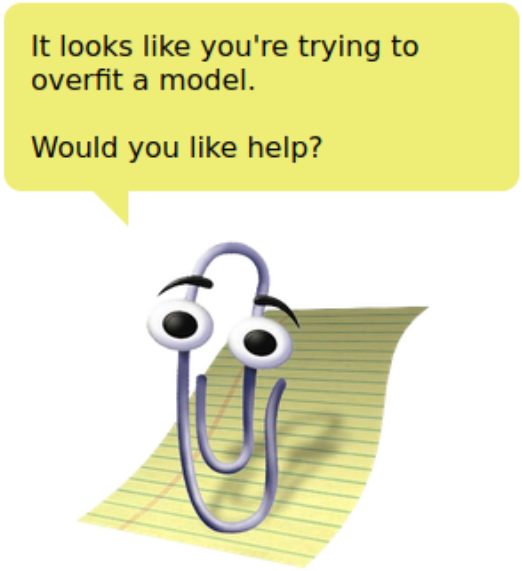


Figure: The OG AI (which was mega annoying by the way).

- (deterministic) linear models as variations of the loss function
- examples of dimensional reduction and feature identifications
- clustering as a way to create features
- need for hyper-parameter tuning and model validation

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

Moving to a Jupyter notebook →