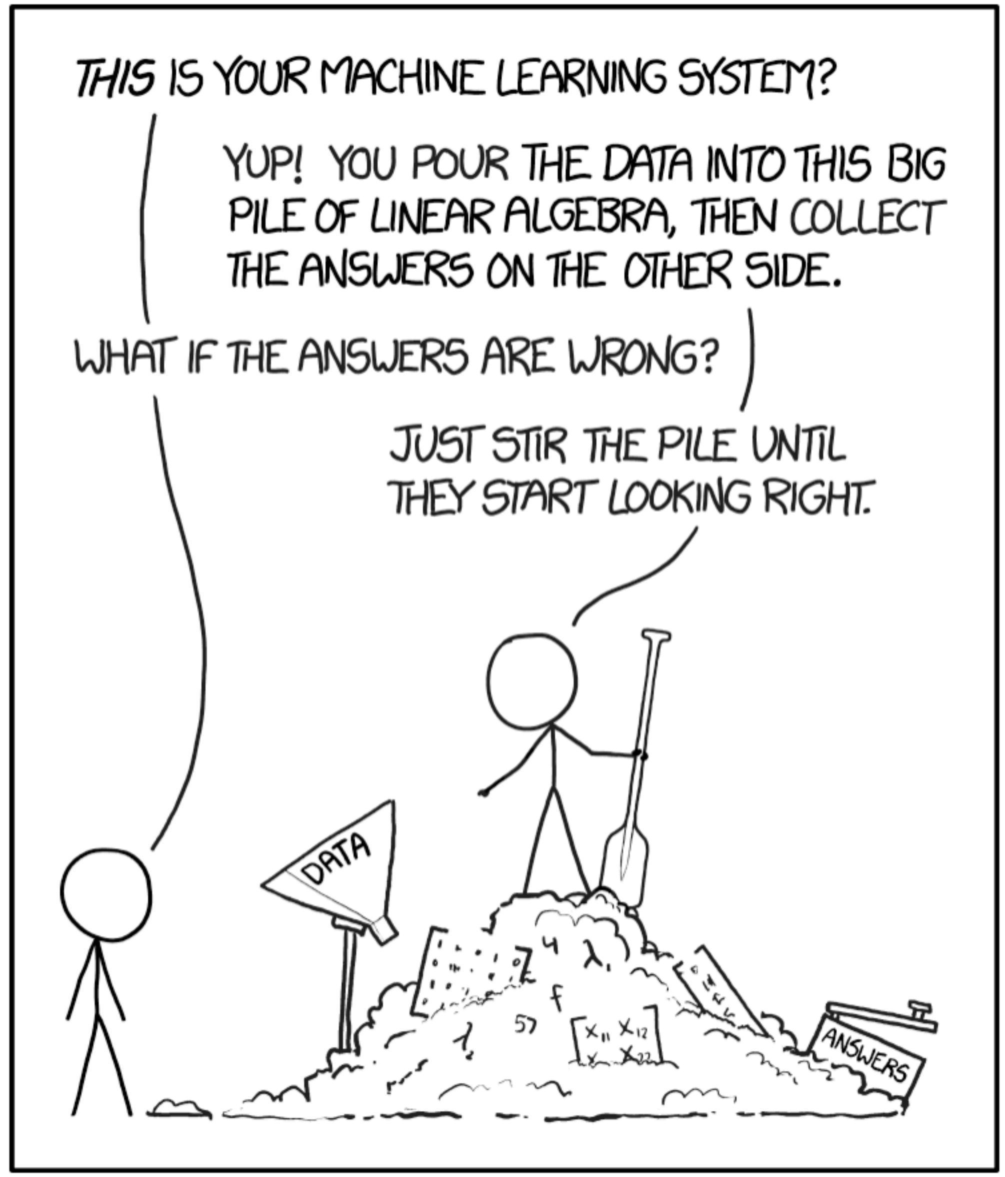
Choices on tests and loss functions



|< < PREV RANDOM NEXT > >|

PERMANENT LINK TO THIS COMIC: HTTPS://XKCD.COM/1838/

Apr.29th 2020

Applied Machine Learning & Big Data Analysis

(Adriano)

This Morning Session

- **Something very quick:** SLACK and workstreams
- **Splitting** choices on the data sets
- **The loss function**: where does it come from? What should I choose?

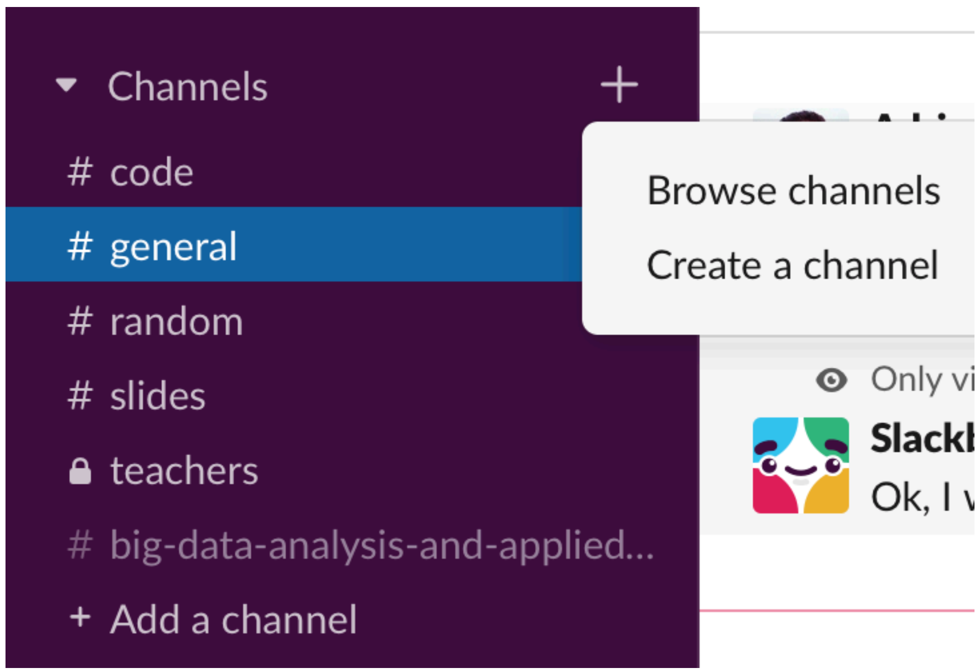
This Afternoon Session

- **Hyperparameter optimisation:**

  All you ever wanted to know but were too ashamed to ask

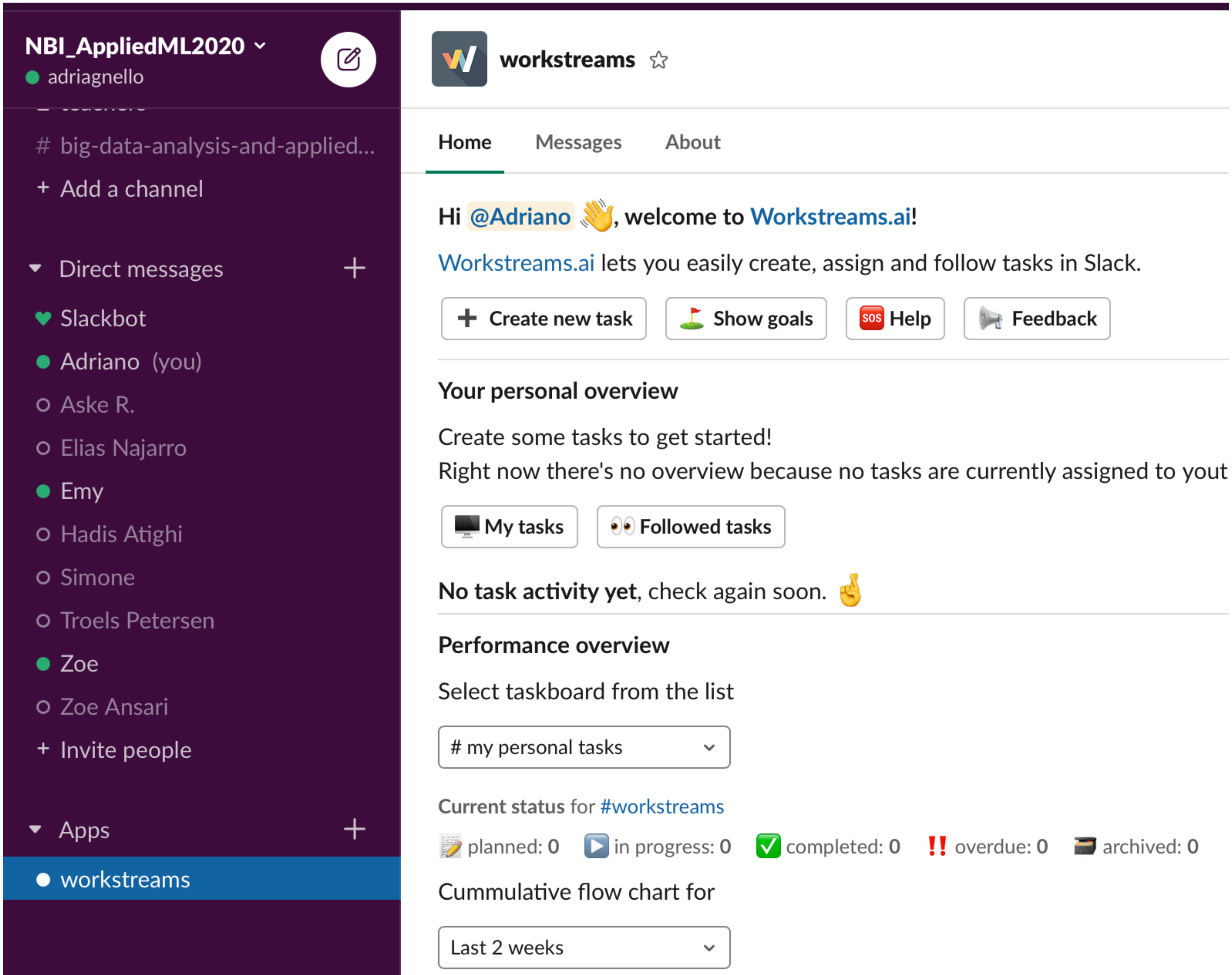
- **Cross-validation**, again

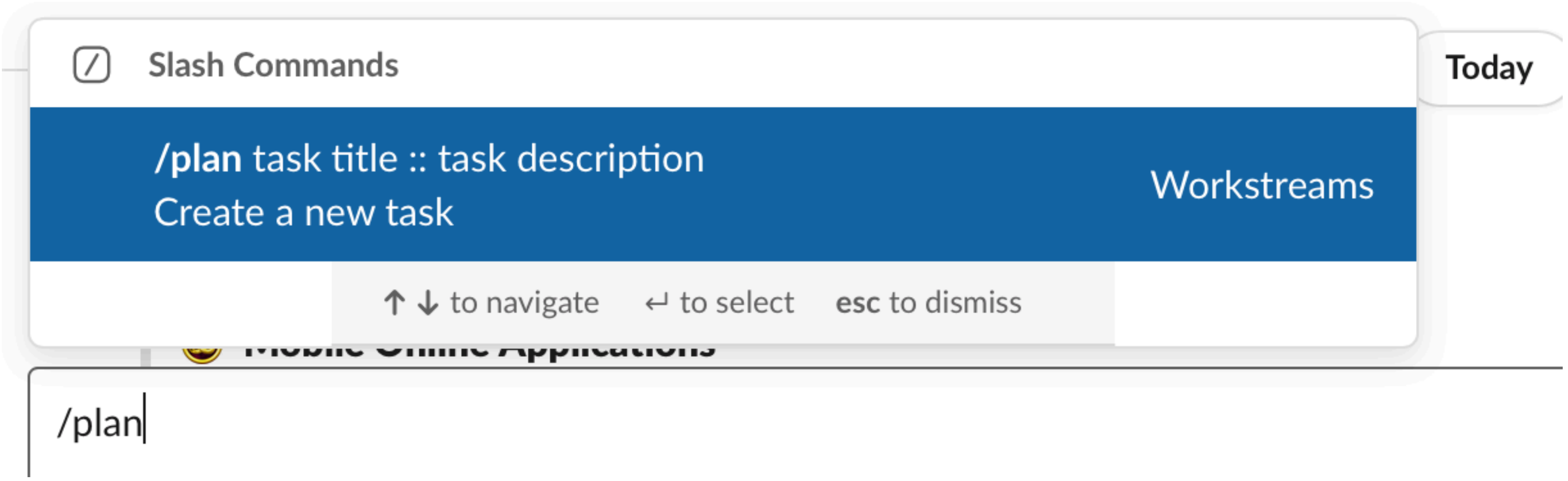
Before we begin: SLACK and Workstreams



You can use this
to create your own channels
for group work (or for book-keeping)

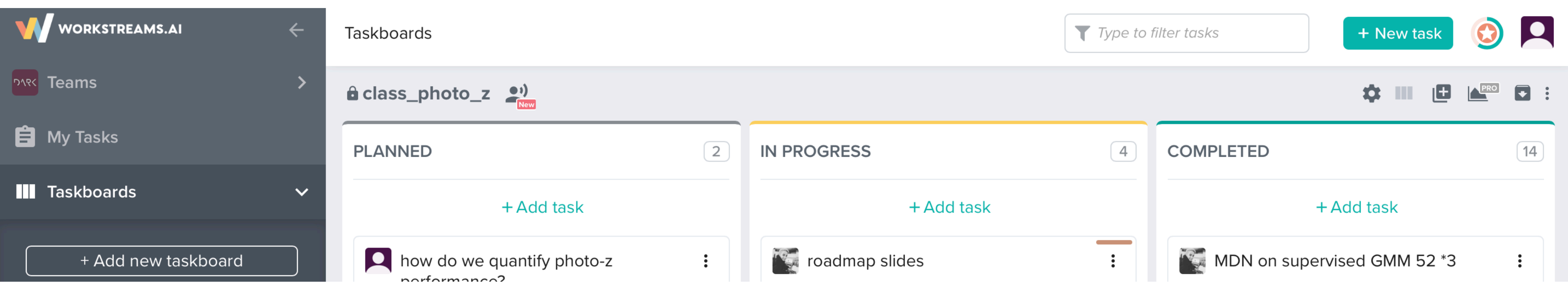
Workstreams



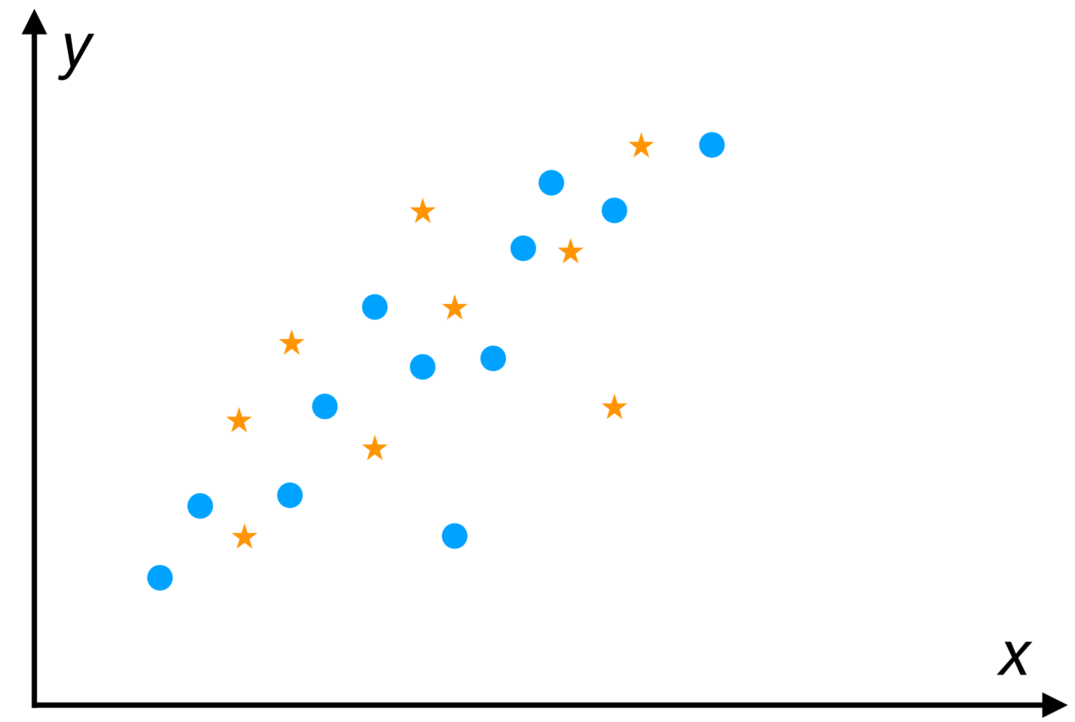


You can use the workstreams dashboard
To issue tasks and track progress
(it's click and drag, really)

You can also issue tasks
With workstreams from within SLACK
(no need to install other stuff)

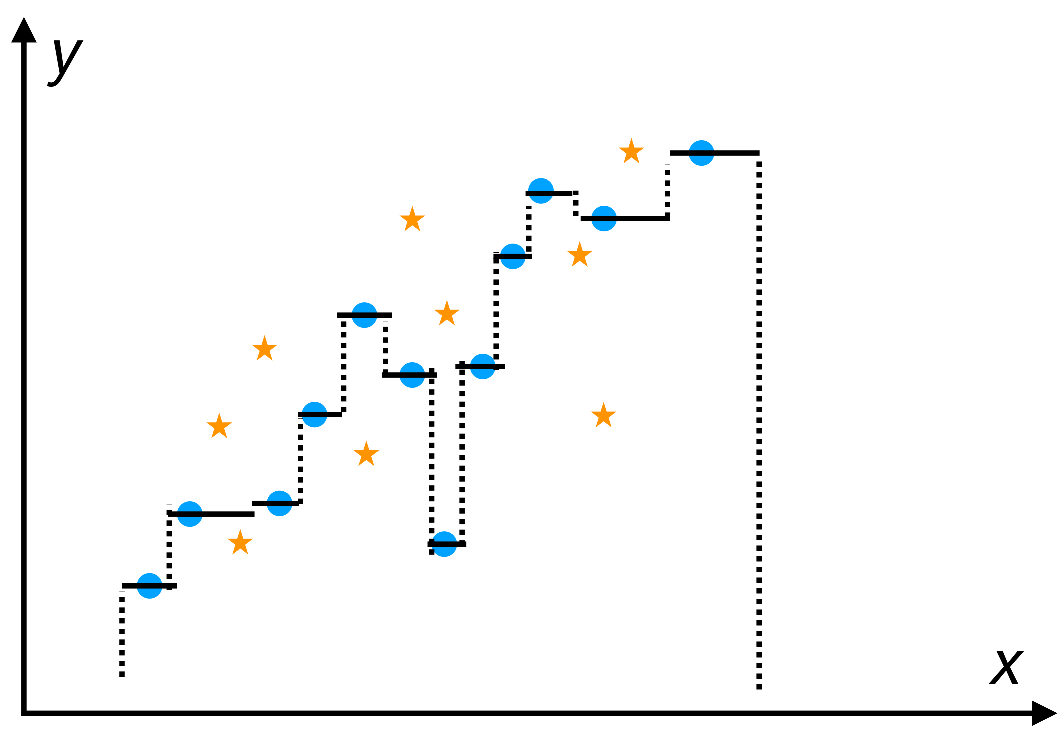


Training and Validating



The data-set:
Objects drawn from some (possibly unknown) distribution, possibly with some outliers…
And we get just some of them to train our machinery.

The issue:
We can push our machinery to fit them perfectly,
But **(a)** does it make sense?
(b) how does it behave once we get more data?



The bias-variance tradeoff

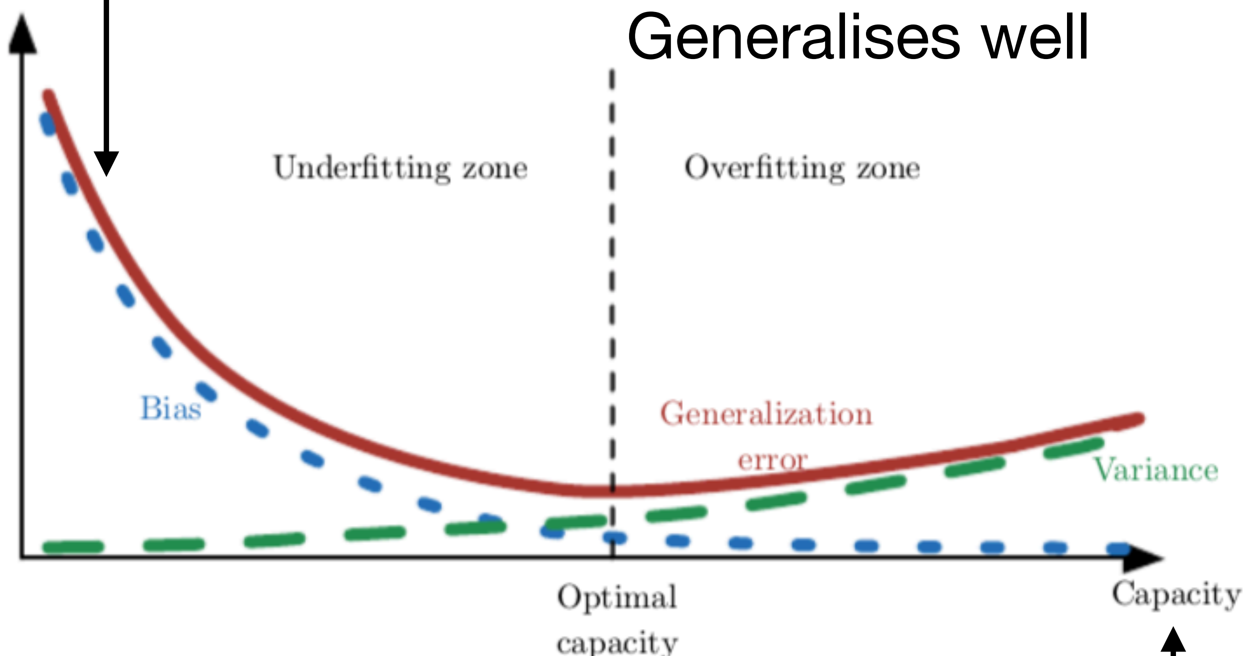
Q: suppose you draw some data and do some least-squares fitting to it,
How can we write the standard deviation (over many draws) of the fitting error?

$$\mathbb{E}[(y - f(x))^2] \ = ?$$

Bias-variance in general

High error on everything

Just right: decent performance,
Generalises well



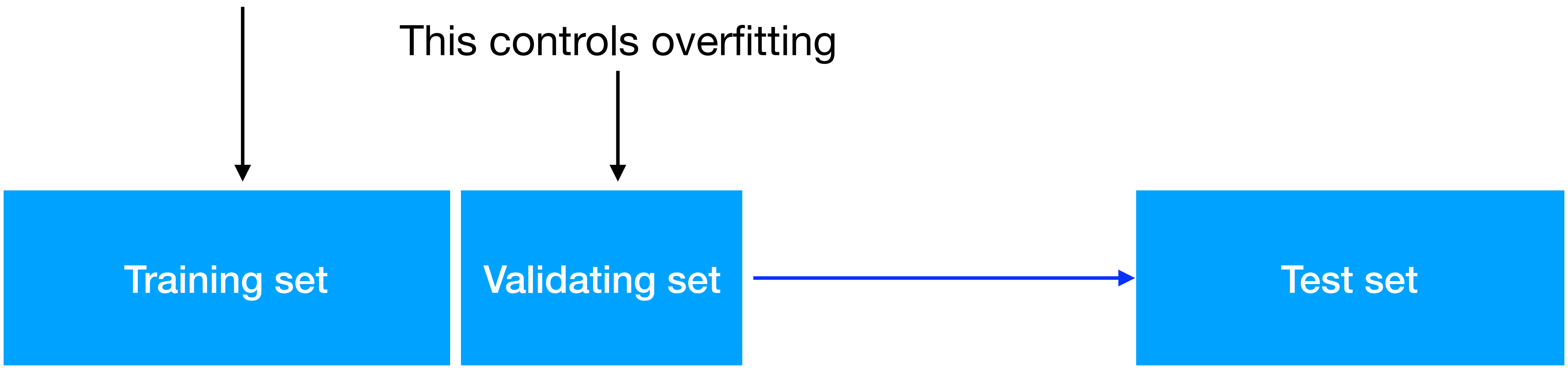
Overfitting: fit better and better,
but generalises badly
when presented with new data

This means: tree depth, NN depth,
NN nodes, ***training epochs***…

Train, Validate, Test

This does the heavy lifting:
Training the machinery

This controls overfitting



Either ***real or simulated***
(anyways labeled) data

The actual data
that you want to work on

Actually: over the last ~5 years
one often trains on the training set
and tests on the test set

```
sklearn.model_selection.train_test_split(*arrays, **options)
```

NB The dataset may not be *balanced*! E.g. one class may be over-sampled.

Early stopping 101

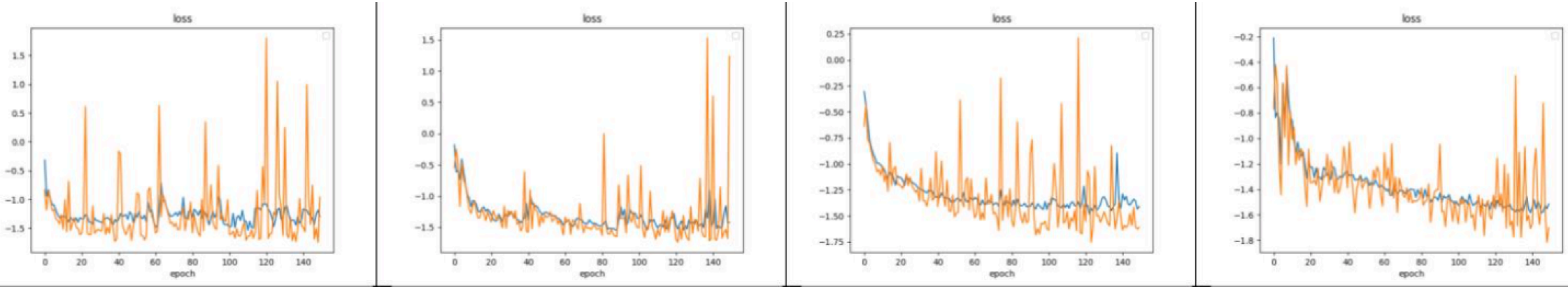
‘patience’=epochs after val.err.minimum

Example in keras
(from Sofie H. Bruun)

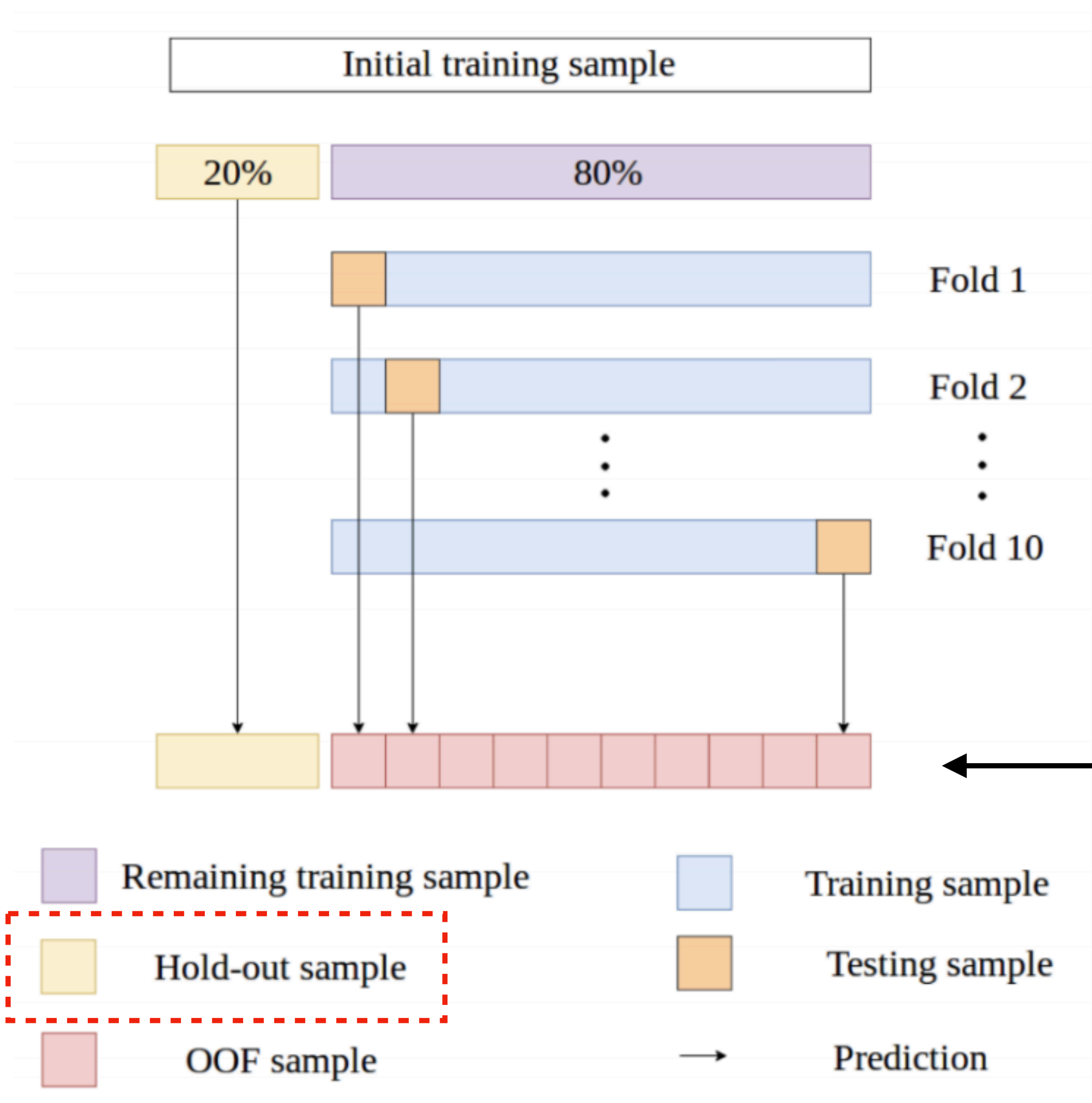
```
callbacks = [tf.keras.callbacks.EarlyStopping(monitor='loss', patience=20)]

model.fit(train_dataset, epochs=2000, verbose=1, validation_data=val_dataset, validation_freq=1, callbacks=callbacks)
```

NB This also depends on the optimiser! Some methods are not a simple gradient descent.

E.g. one word of caution…
(courtesy of Zoe, ongoing project)

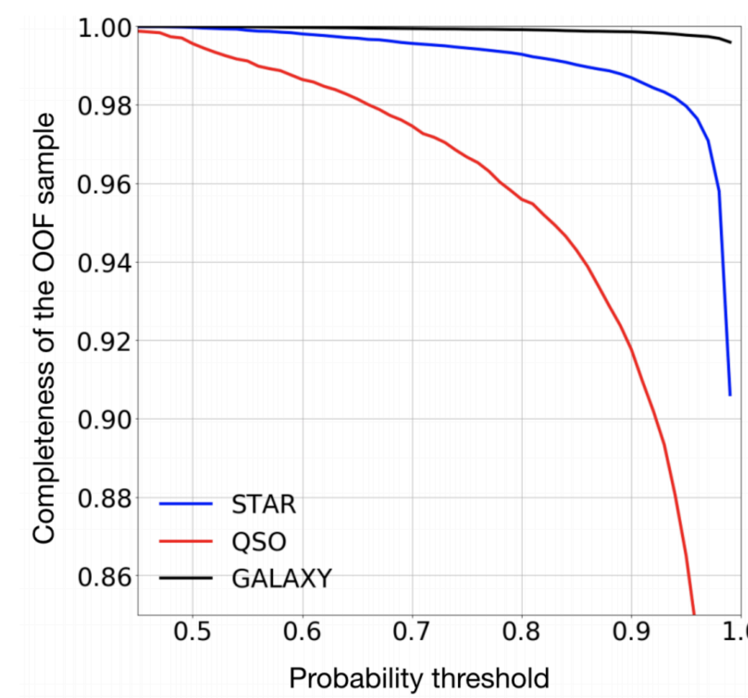


Cross-Validation 101



(source: [this paper](#))

```
Pseudocode for cross-validation
all_folds = split_into_k_parts(all_training_data)

for set_p in hyperparameter_sets:
    model = InstanceFromModelFamily()

    for fold_k in all_folds:
        training_folds = all_folds besides fold_k
        fit model on training_folds using set_p
        fold_k_performance = evaluate model on fold_k

    set_p_performance = average all k fold_k_performances for set_p

select set from hyperparameter_sets with best set_p_performance
```



Bonus track: out-of-bag estimates:
https://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_oob.html

Loss Functions: WHYs and HOWs, 1

Least-squares:

$$\text{loss} \;=\; \sum_{i=1}^{N} \frac{(y_i - f(x_i))^2}{\sigma^2 + \epsilon_i^2}$$

(you may have often seen it without the denominators, and with a *1/N* in front of it)
- easy to operate with (smooth derivatives)
- comes from Gaussian statistics

$$\mathrm{p}(y_i\,|\,x_i) \;=\; \mathcal{G}\left(f(x_i),\,\sqrt{\sigma^2+\epsilon_i^2}\right), \quad \text{loss} \sim -\log(\prod_i \mathrm{p}_i) + \text{cnst}$$

Issues and variations:
- What if we are not drawing data from Gaussian distributions?
- What about fat tails and outliers?
- Alternative choice: **MAD**

$$\text{loss} \;=\; \sum_{i=1}^{N} \frac{|\,y_i - f(x_i)\,|}{\sqrt{\sigma^2 + \epsilon_i^2}}$$

Plus: Less sensitive to large deviations. Minus: derivative at zero???
Q: how would you write the loss function for power-law distributions

$$\mathrm{p}(y_i\,|\,x_i) \;=\; \frac{\nu(\alpha)/\epsilon_i}{\left(1 + (y_i - f(x_i))^2/(\alpha\epsilon_i^2)\right)^{\alpha/2}}$$

Loss Functions: WHYs and HOWs, 2

Cross-entropy?

$$\text{loss} \;=\; -\sum_{i=1}^{N} y_i \log f(x_i) + (1-y_i)\log(1-f(x_i))$$

- comes from Bernoulli statistics

$$\mathrm{p}(y_i\,|\,x_i) \;\sim\; f(x_i)^{y_i}(1-f(x_i))^{1-y_i}$$

Q: is this a good way to tackle multi-class problems?

- Q1: think about given class labels, and predictions for each object (e.g. ANN)
- Q2: think about given class labels, but rougher estimates (e.g. trees)

- Q3: if I use gradient descent, is the best-fit *f* an unbiased predictor of *y*?
  (hint: Laplace's *rule of succession*)

Q: if you use Gaussians as in the previous slides, what is the best-fit estimator of sigma? What is the *unbiased* estimator of sigma?

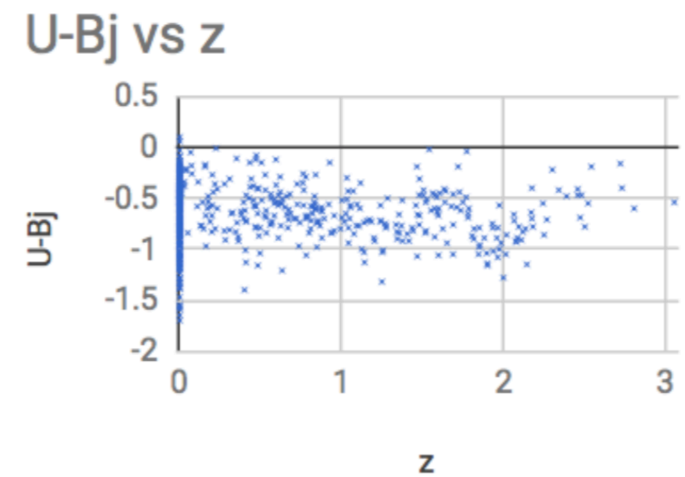
Loss Functions: WHYs and HOWs, 3

Mixture Density Networks (MDNs)

Very nice introduction **here, with code**

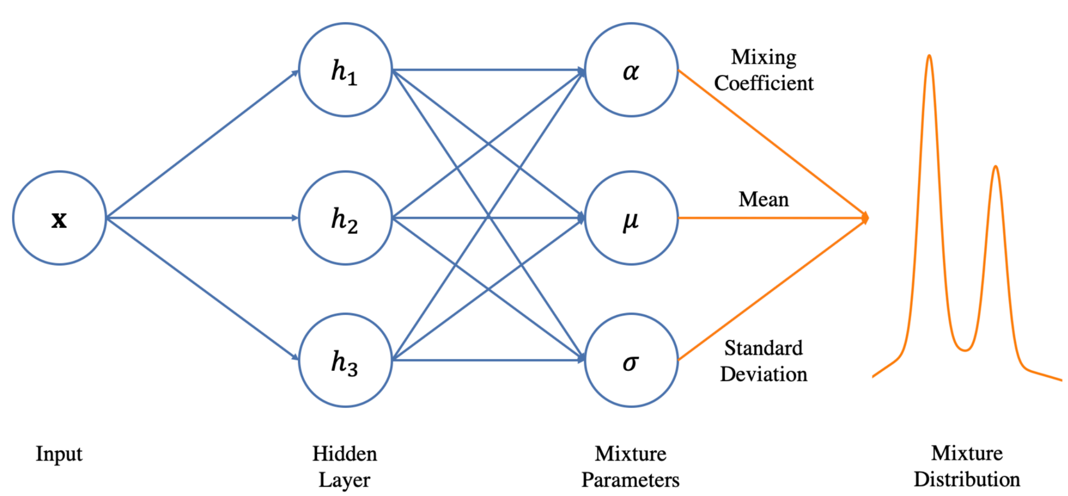
What if multiple values of *y* can correspond to the same *x*?

Example/Exercise:
in the 'QSO' objects from Monday's exercise,
plot *y*=redshift vs *x*=g-r



Likelihoods: $$p(y_i|x_i) \sim \sum_{k=1}^{K} w_k(x_i)\mathcal{G}(\mu_k(x_i), \sigma_k(x_i))$$

Loss function: $$\text{loss} = -\sum_{i=1}^{N} \log(p(y_i|x_i)) = -\sum_{i=1}^{N} \log\left(\sum_{k=1}^{K} \hat{w}_k \mathcal{G}(\hat{\mu}_k, \hat{\sigma}_k)\right)$$



HARD Exercise:
in the 'QSO' objects from Monday's exercise, how well can you predict redshift using the *SDSS* magnitudes?