Advanced Python for Neuroscientists

Summer 2021

Princeton Neuroscience Institute
Instructors: Yisi Zhang & Tyler Giallanza

July 6, 2021

Contact

- Yisi Zhang: yz9@princeton.edu
- Tyler Giallanza: tylerg@princeton.edu

Schedule

Tue:	2-3:30	Lecture
Thu:	4-5:30	Lecture
Fri:	1-2:30	Precept & office hour

Preview

- Learning from data
- Regression
- Classification
- Unsupervised learning
- Neural networks I
- Neural networks II
- Deep learning
- Reinforcement learning

Set up

1. Google “colab”
2. Upload colab notebook
3. Done!

Lecture 1: Learning from data

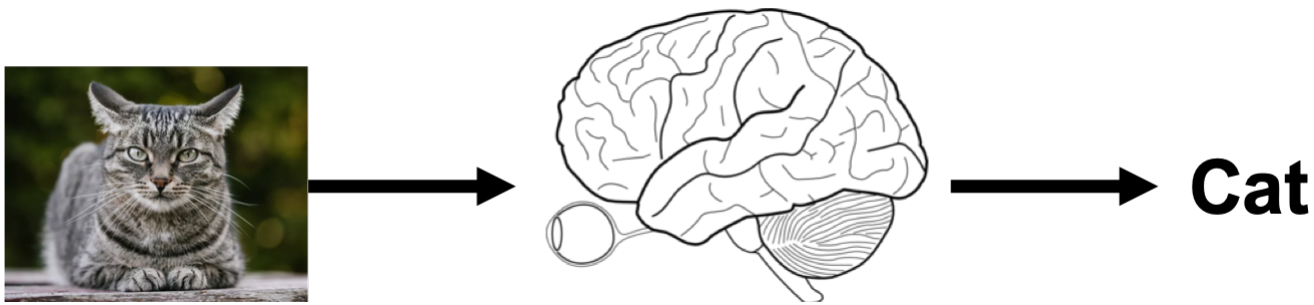
Outline

- Machine learning in neuroscience
- The accuracy-interpretability trade-off
- The bias-variance trade-off
- Training vs. testing
- Supervised vs. unsupervised learning

1.1 Machine learning in neuroscience

Example 1

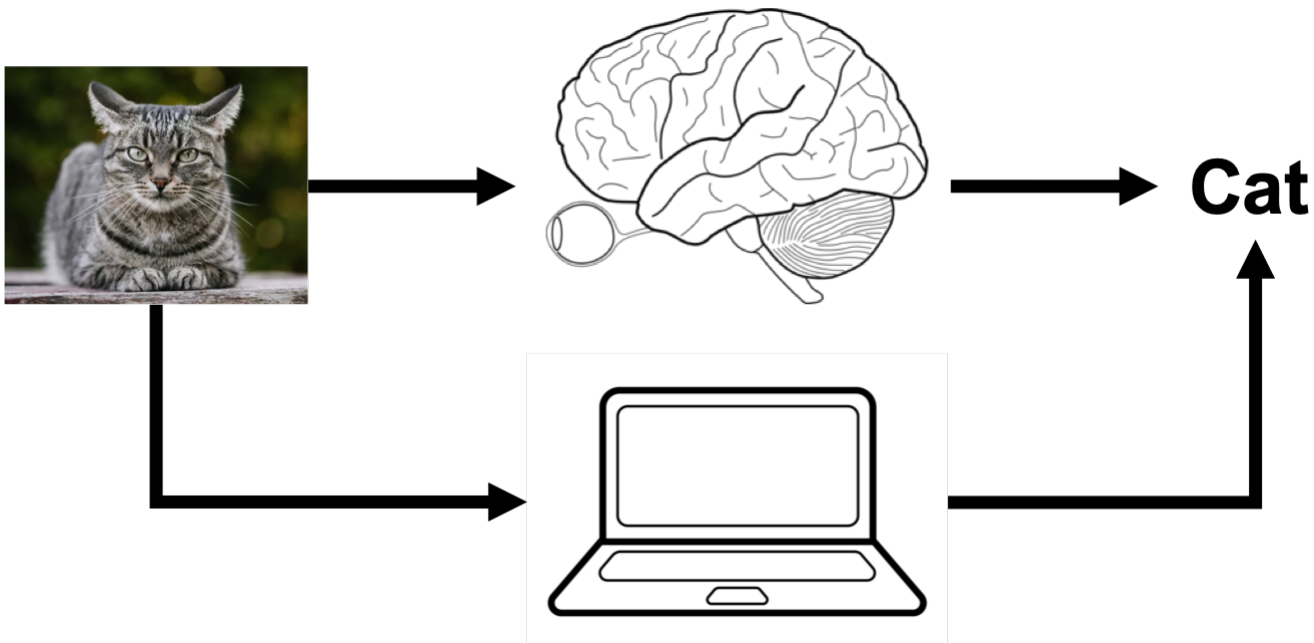
$f_{brain}(X)$



1.1 Machine learning in neuroscience

Example 1

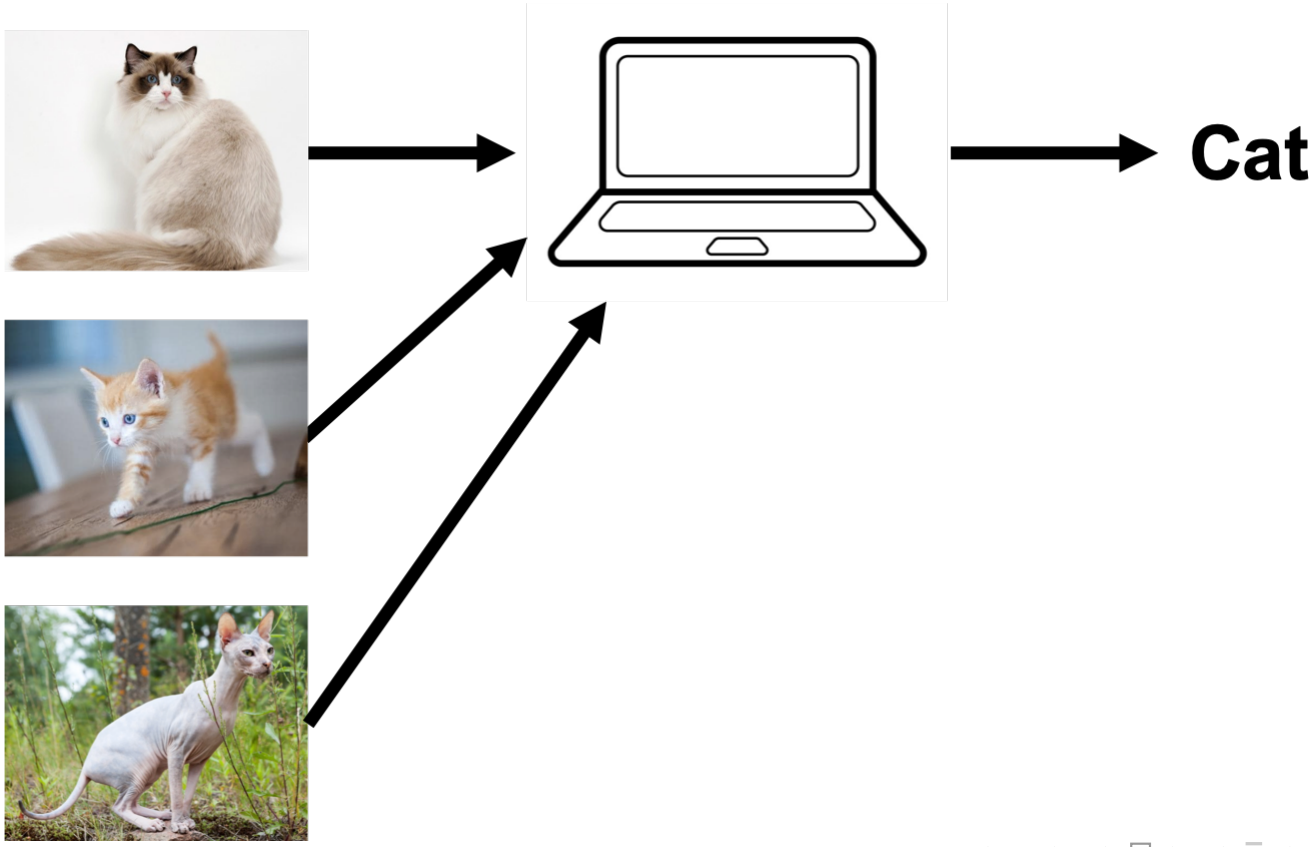
$$f_{brain}(X)$$



$$f_{machine}(X)$$

1.1 Machine learning in neuroscience

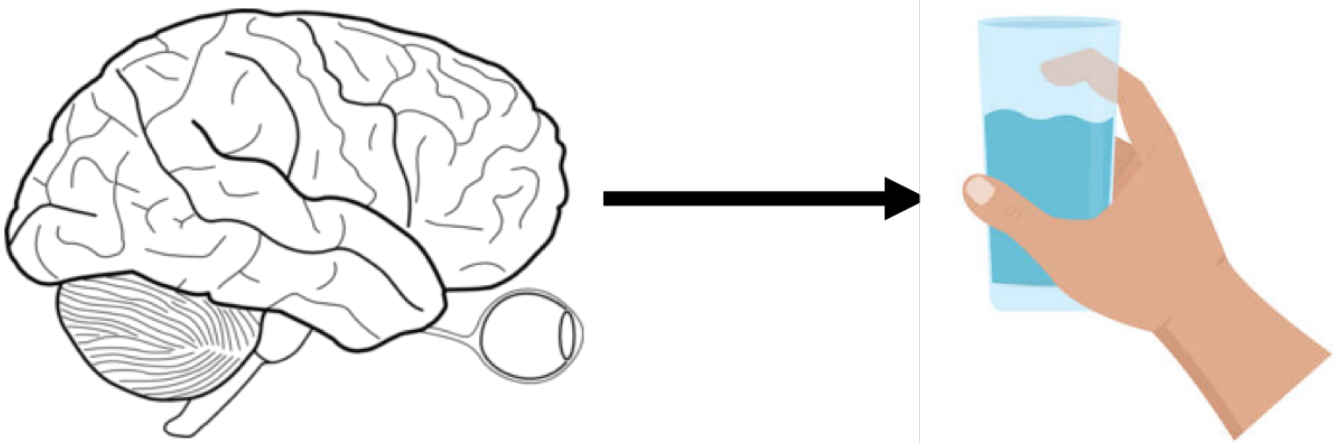
Example 1



1.1 Machine learning in neuroscience

Example 2

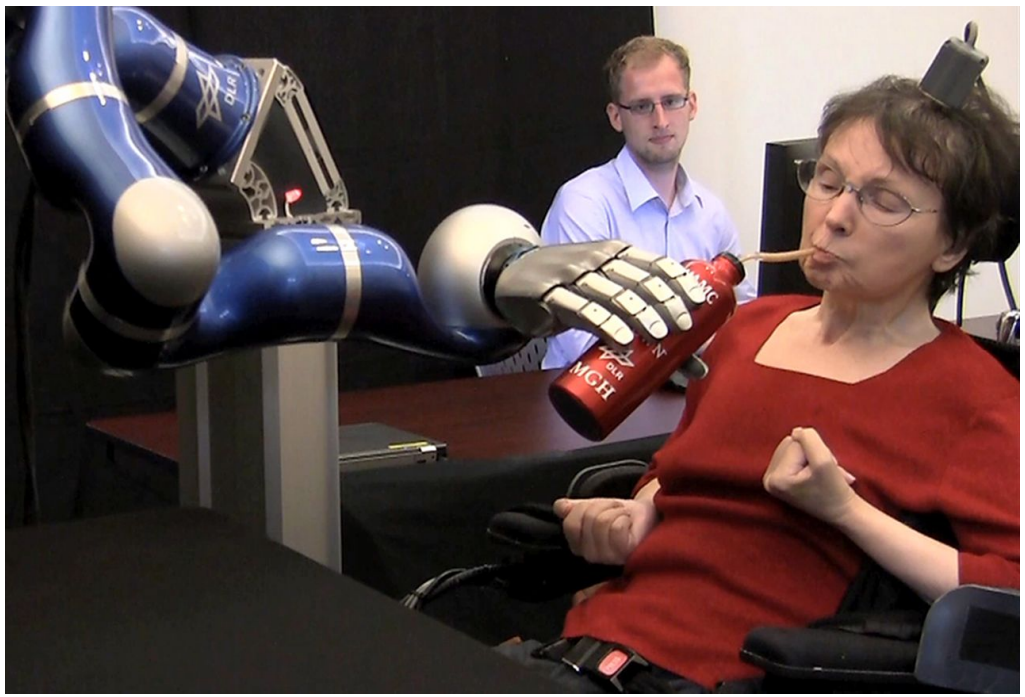
$$Y_{muscle} = f(X_{neuro})$$



1.1 Machine learning in neuroscience

Example 2

$$Y_{robot} = \tilde{f}(X_{neuro})$$



Prashant Nair "Brain–machine interface" *PNAS* 2013

Can you come up with an example?

How to find f?

Suppose that we have an observation Y and p different predictors $X_1, X_2, ..., X_p$. We assume that there is some relationship between Y and $X = (X_1, X_2, ..., X_p)$

$$Y = f(X) + \epsilon,$$

where f is an unknown function of $X_1, X_2, ...X_p$, and ϵ is a random error that is independent of X and has mean zero.
A major task of machine learning is to find a proper f.

How to find f?

"Essentially, all models are wrong, but some are useful." — Empirical Model Building and Response Surfaces, Box G and Draper N

"Everything should be made as simple as possible, but no simpler." — Albert Einstein

A proper model should

Fit the data well, predict future observations, be somewhat interpretable,...

1.2 The accuracy-interpretability trade-off

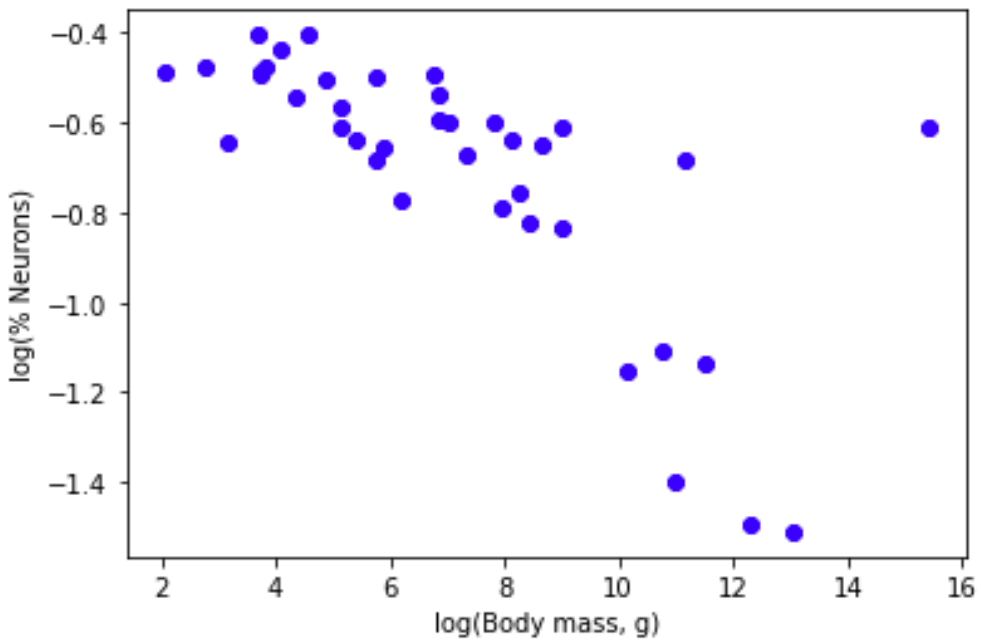
How many neurons does a brain have? – A toy example

	Species	Order	n	Body mass, g	Brain mass, g	Neurons	Other cells	% Neurons	Source
0	Sorex fumeus	Eulipotyphla	3	7.8	0.176	36460000	22860000	61.3	Sarko et al., 2009
1	Blarina brevicauda	Eulipotyphla	5	16.2	0.347	55190000	33550000	64.8	Sarko et al., 2009
2	Heterocephalus glaber	Glires	3	23.3	0.392	26875462	24191205	52.5	Herculano-Houzel et al., 2011
3	Mus musculus	Glires	4	40.4	0.402	67873741	33858759	65.3	Herculano-Houzel et al., 2006
4	Parascalops breweri	Eulipotyphla	3	42.7	0.759	123600000	78260000	61.2	Sarko et al., 2009

Suzana Herculano-Houzel "Mammalian brains are made of these" *BBE* 2015

1.2 The accuracy-interpretability trade-off

Predict % neurons among all cell types in the brain by body mass
$y = log(\%neurons)$, and $x = log(body\ mass)$



What would be a good model for $y = f(x)$?

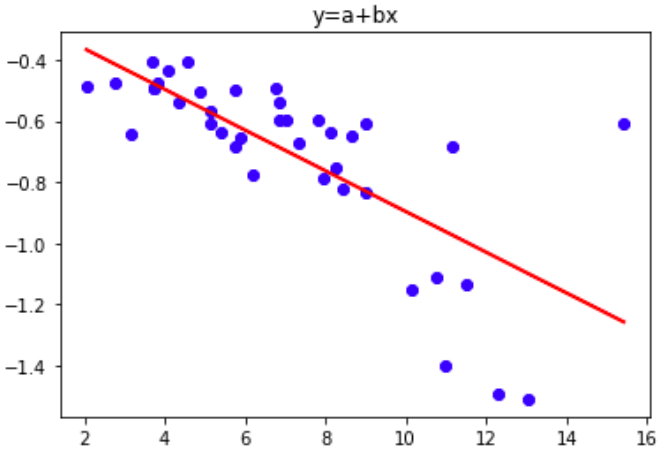
1.2 The accuracy-interpretability trade-off

Exercise: Prepare X and Y for model fitting

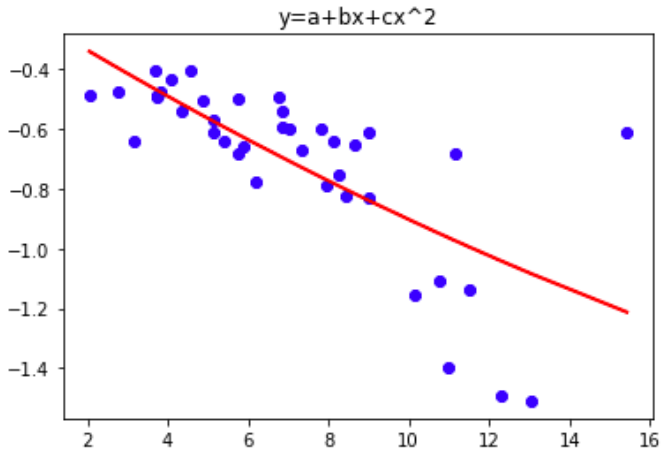
```
import numpy as np
import pandas as pd
# Read csv file into a pandas dataframe
 url = 'https://raw.githubusercontent.com/
yisiszhang/AdvancedPython/main/data/wholebrain_neuron.csv'
df = pd.read_csv(url, thousands=',')
x = np.array(np.log(df['Body mass, g']))
y = np.array(np.log(df['Neurons']/(df['Neurons'] + df['Other
cells'])))
x = x.reshape(-1,1)
# what does reshape(-1,1) do here?
```

1.2 The accuracy-interpretability trade-off

Which model is better?




```
Coefficients:
 [[-0.06659798]]
Mean squared error: 0.0382
Coefficient of determination: 0.52
```




```
Coefficients:
 [[-0.08301135  0.00101804]]
Mean squared error: 0.0380
Coefficient of determination: 0.53
```

The prediction error is marginally improved with the quadratic model, so the linear model is more suitable and easier to interpret as $\%neurons = e^a bodymass^b$, i.e., a power-law relationship. This may help us gain some insights.

1.3 The bias-variance trade-off

To evaluate the performance of a model, we usually quantify the accuracy of the predicted response compared to the true value. In regression setting, the most commonly used measure is the *mean squared error* (MSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

$\hat{f}$ is an estimate for f and $\hat{f}(x_i)$ is the prediction that $\hat{f}$ gives for the ith observation.
Many statistical methods estimate coefficients to minimize the MSE.

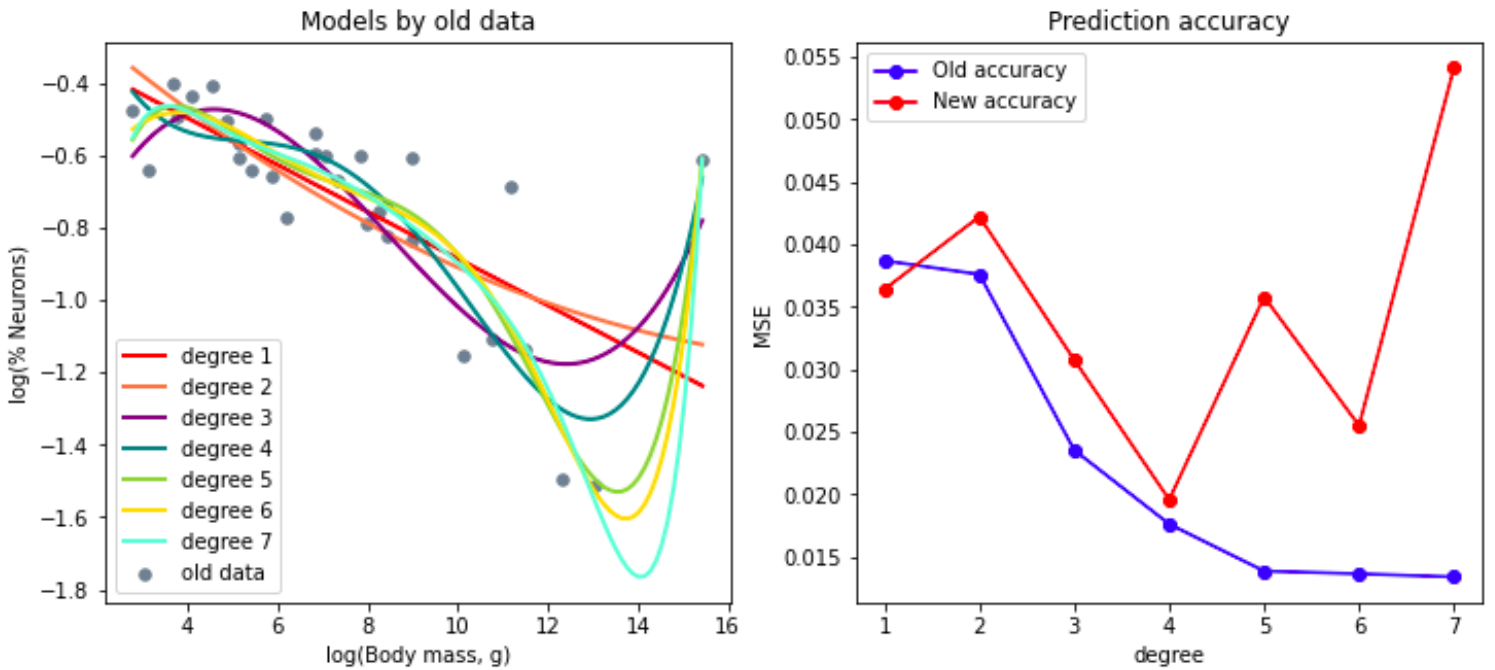
1.3 The bias-variance trade-off

Now, if we collect some new data, can the model fitted by the old data well predict our new data?

More complex models usually give lower MSE (unless the number of model parameters $p > N$).
What if we achieved a 0.000000001 MSE (relative to the scale of the data) using a model fitted by our old data?

1.3 The bias-variance trade-off

Usually, the insanely accurate models DO NOT work well for new data.



1.3 The bias-variance trade-off

Exercise: Polynomial variables

```
from sklearn.preprocessing import PolynomialFeatures
# For y = a + bx + cx^2, we need 1, x, x**2
poly = PolynomialFeatures(2)
x_poly = poly.fit_transform(x)
# This equivalent to
# x_poly=np.hstack((np.ones_like(x), x, x**2))
```

1.3 The bias-variance trade-off

For a given x_0, the expected MSE is

$$E(y_0 - \hat{f}(x_0))^2 = \text{Var}(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\epsilon).$$

The *variance* of $\hat{f}$ usually increases with the number of parameters in the model. The *bias* is created by approximating a real-life problem with a somewhat simplified model.
Usually, the error vs. model complexity curve is "U-shaped".
There are methods to find an optimal model (we will cover in next two lectures).

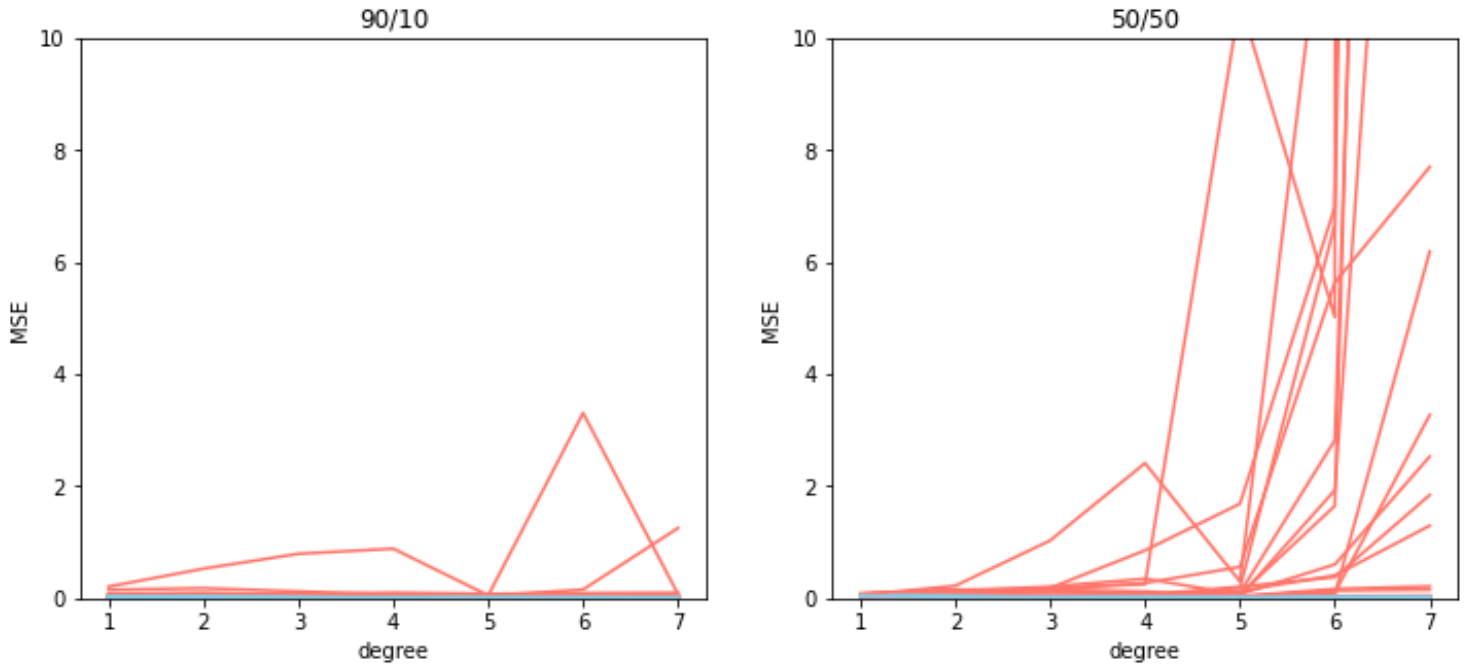
1.4 Training vs. testing

It is good practice to set aside a random subset of the data as the test set, and make sure the information of the training set does not sneak into the test set through, e.g., normalization.
What should we consider when splitting data?

More training data give us more accurate estimate of the model.
More test data give us more accurate estimate of the prediction error.
The dependence on the choice of partition is sensitive if sample size is small.

1.4 Training vs. testing

Our example has a small sample size of $N = 39$. The models trained with 50/50 split suffer from overfitting easily (Why?).



A 80/20 split is in general a good start.

1.4 Training vs. testing

Exercise: Split data into training and test sets

```
# Here is an example to define a function
 def   split_train_test(data, test_ratio):
       shuffled_indices = np.random.permutation(len(data))
       test_set_size = int(len(data) * test_ratio)
       test_indices = shuffled_indices[:test_set_size]
       train_indices = shuffled_indices[test_set_size:]
       return data.iloc[train_indices], data.iloc[test_indices]
# Or cheat
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(df, test_size=0.2)
```

1.5 Supervised vs. unsupervised learning

So far, we have discussed situations that fall into the supervised learning domain. For each predictor x_i, there is an associated response y_i. The aim of our model fitting is accurately predicting the response for future observations (prediction) or better understanding the relationship between the response and the predictors (inference).

1.5 Supervised vs. unsupervised learning

Variables can be characterized as
Quantitative or numerical:

- Discrete (e.g. number of neurons, number of spikes)
- Continuous (e.g. brain mass, firing rate)

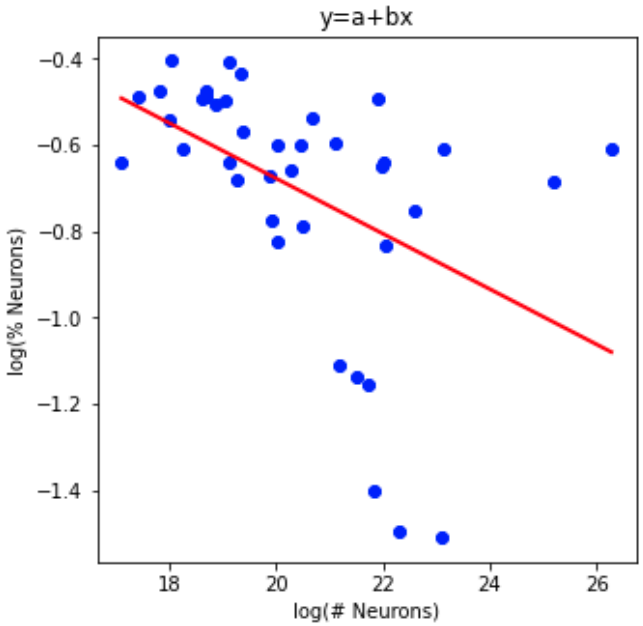
Qualitative or categorical:

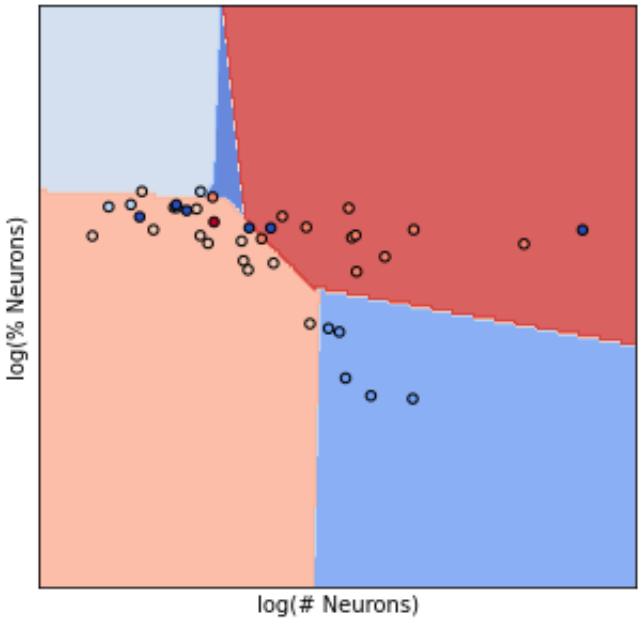
- Unordered (e.g. species, cell type)
- Ordered (e.g. developmental stages)

We tend to refer to problems with a quantitative response as regression problems, and those with a qualitative response as classification problems.

1.5 Supervised vs. unsupervised learning

Regression vs. classification





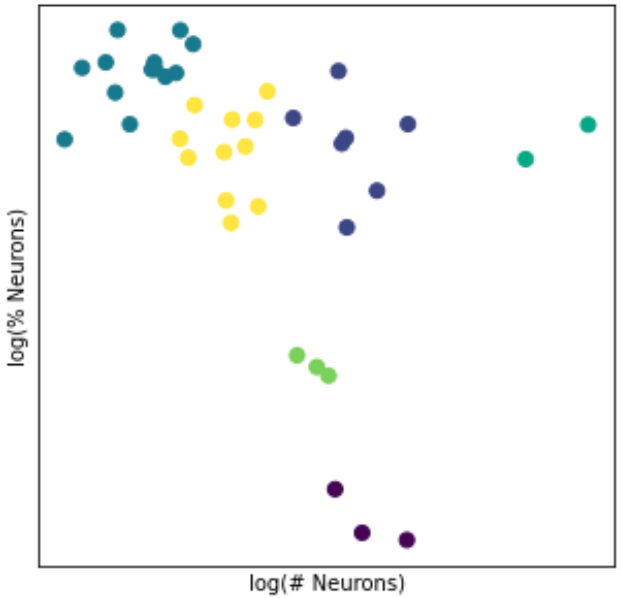
Depending on the goal, we may either find the relationship between two quantities or determine the quantitative boundaries for classification.

1.5 Supervised vs. unsupervised learning

Unsupervised learning deals with situations where the data does not contain any response or output information. How can we learning anything from mere inputs?

1.5 Supervised vs. unsupervised learning

Now if we are not given the information of the species, we find that the *unlabeled data* show characteristics in different regions of the feature space not necessarily overlapping with the partitions by species.



Problems like this are referred to as clustering problems.

1.5 Supervised vs. unsupervised learning

Exercise: Make X and y for classification

```
# stack two variables
x1 = np.array(df['log Neurons'])
x2 = np.array(df['log % Neurons'])
x1 = x1.reshape(-1,1)
x2 = x2.reshape(-1,1)
X = np.hstack((x1, x2))
# convert y to categorical data, use numerical code
y = df['Order '].astype("category").cat.codes
```

Homework

- Make sure you understand all the exercises above
- Run through the codes here that should replicate all the figures
  https://github.com/yisiszhang/AdvancedPython/blob/main/colab/Lecture1.ipynb
- Data splitting. Try with different random number seeds and splitting strategies to see if you get consistent results. Decide how you would split the data.
- Decide which polynomial model to use and write down your rationale.