# Introduction to machine-learning for neuroimaging

May 2024

*By*

*Qing Wang* (Vincent)

王庆

https://github.com/Vincent-wq

国家精神疾病医学中心
NATIONAL CENTER FOR MENTAL DISORDERS
上海市精神卫生中心
SHANGHAI MENTAL HEALTH CENTER

neuro
Montreal Neurological Institute-Hospital

# Contents

- Part 1: Machine Learning Basics
  - Describe basics of the “learning” process
  - Explain model design choices and performance trade-offs
  - Model selection and validation frameworks
- Part 2: Machine Learning for Neuroimaging
  - Python lib family for neuroimaging studies
  - *Coding example: Autism fMRI classification*

*What is machine learning?*

*What is machine learning?*

***Quick Internet search****: It is a branch of artificial intelligence (AI) and computer science that focuses on the using data and algorithms to enable AI to imitate the way that humans learn.*

*What is machine learning?*

***Quick Internet search**: It is a branch of artificial intelligence (AI) and computer science that focuses on the using data and algorithms to enable AI to imitate the way that humans learn*

***Philosophy professor:** You do not control what you make but you control the process of its creation.*

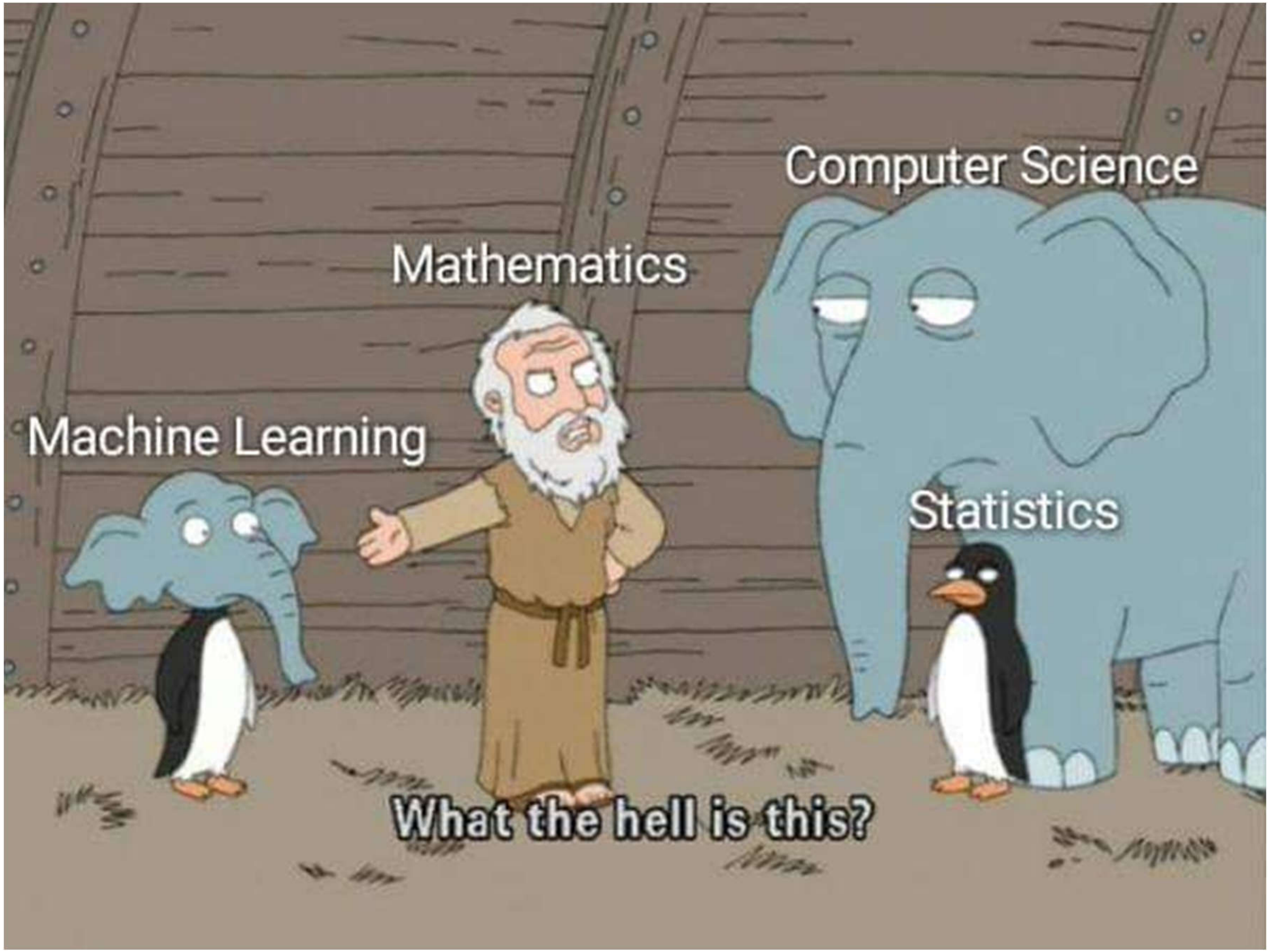
*What is machine learning?*

***Quick Internet search****: It is a branch of artificial intelligence (AI) and computer science that focuses on the using data and algorithms to enable AI to imitate the way that humans learn.*

***Philosophy professor:*** *You do not control what you make but you control the process of its creation.*

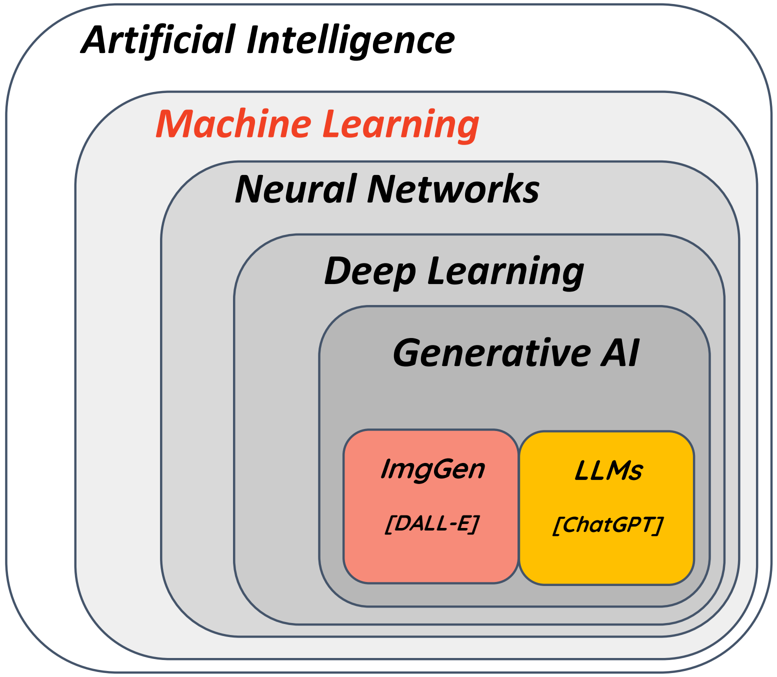
***ChatGPT:*** It is is a type of AI that allows software applications to become more accurate at predicting outcomes without being explicitly programmed. It is based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

*In practice…*


Computer Science
Mathematics
Machine Learning
Statistics
What the hell is this?

# Machine-learning - what, why, and when?

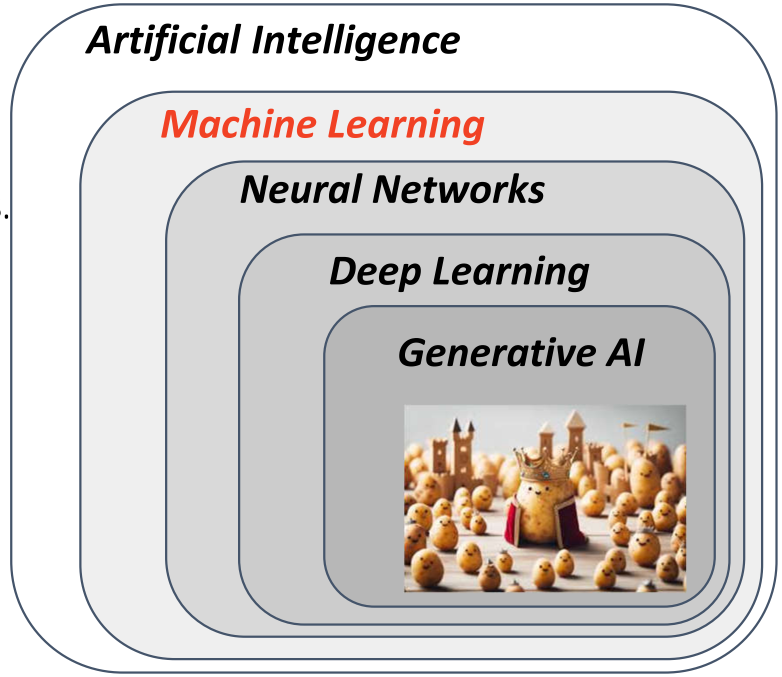
- What is Machine learning (ML)?
    - ML is the study of computer algorithms that improve automatically through experience and by the use of data.



# Machine-learning - what, why, and when?

- What is Machine learning (ML)?
  - ML is the study of computer algorithms that improve automatically through experience and by the use of data.

- Why is it useful - especially in life sciences?
  - Biology, Medicine, Environmental sciences comprise phenomenons (e.g. a disease) with large number of variables.
  - We want to model complex relationships within these variables and make accurate predictions.





9

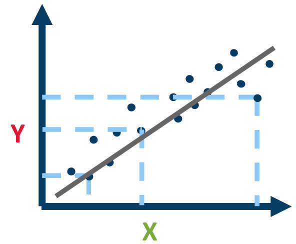
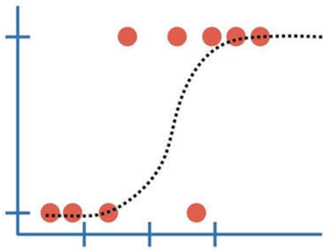
# Machine-learning - what, why, and when?

- What is Machine learning (ML)?
  - ML is the study of computer algorithms that improve automatically through experience and by the use of data.

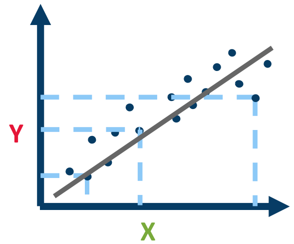
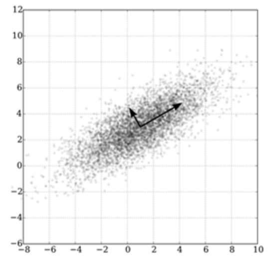
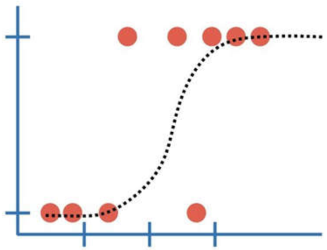
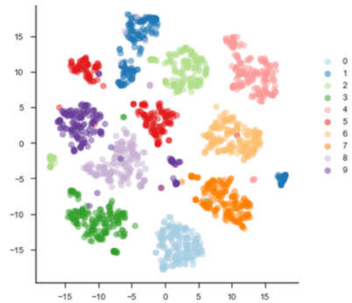
- Why is it useful - especially in life sciences?
  - Biology, Medicine, Environmental sciences comprise phenomenons (e.g. a disease) with large number of variables.
  - We want to model complex relationships within these variables.

- When do I use it?
  - You are interested in 1) prediction tasks or 2) low-dimensional representation.
  - You have sufficient data.

# Types of ML Algorithms

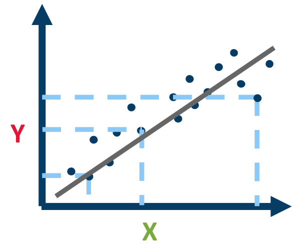
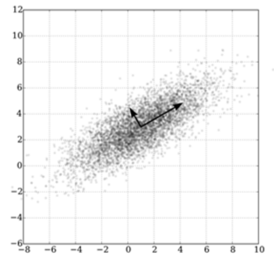
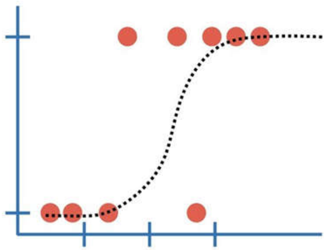
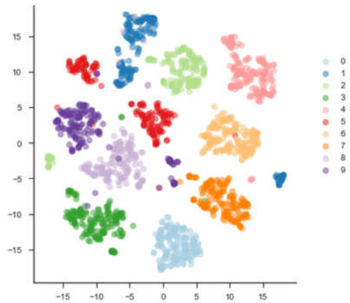
| Outcome | Supervised Learning | |
|---|---|---|
| Continuous | Regression  | |
| Categorical | Classification  | |

# Types of ML Algorithms

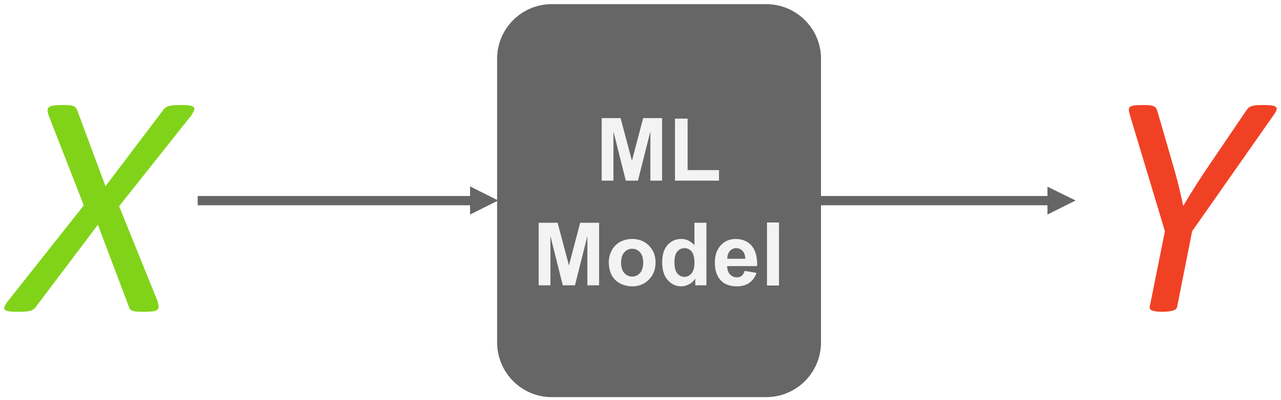
| Outcome | Supervised Learning | Unsupervised Learning |
|---|---|---|
| **Continuous** | Regression   | Principal Component  |
| **Categorical** | Classification  | Clustering  |

**… and Reinforcement Learning*

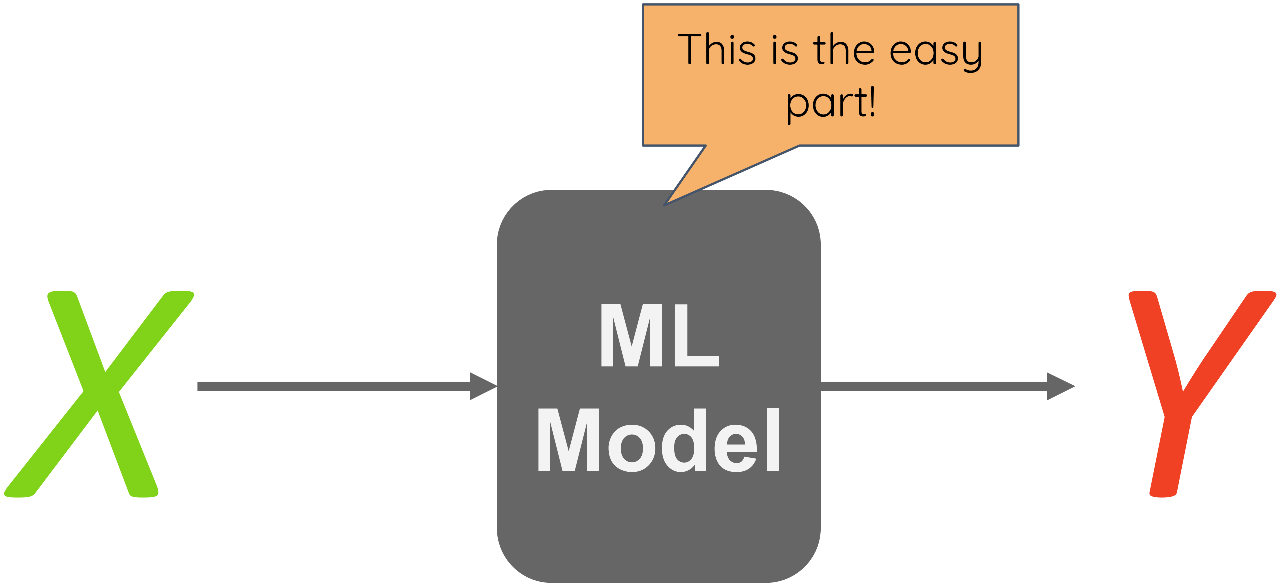
# Types of ML Algorithms

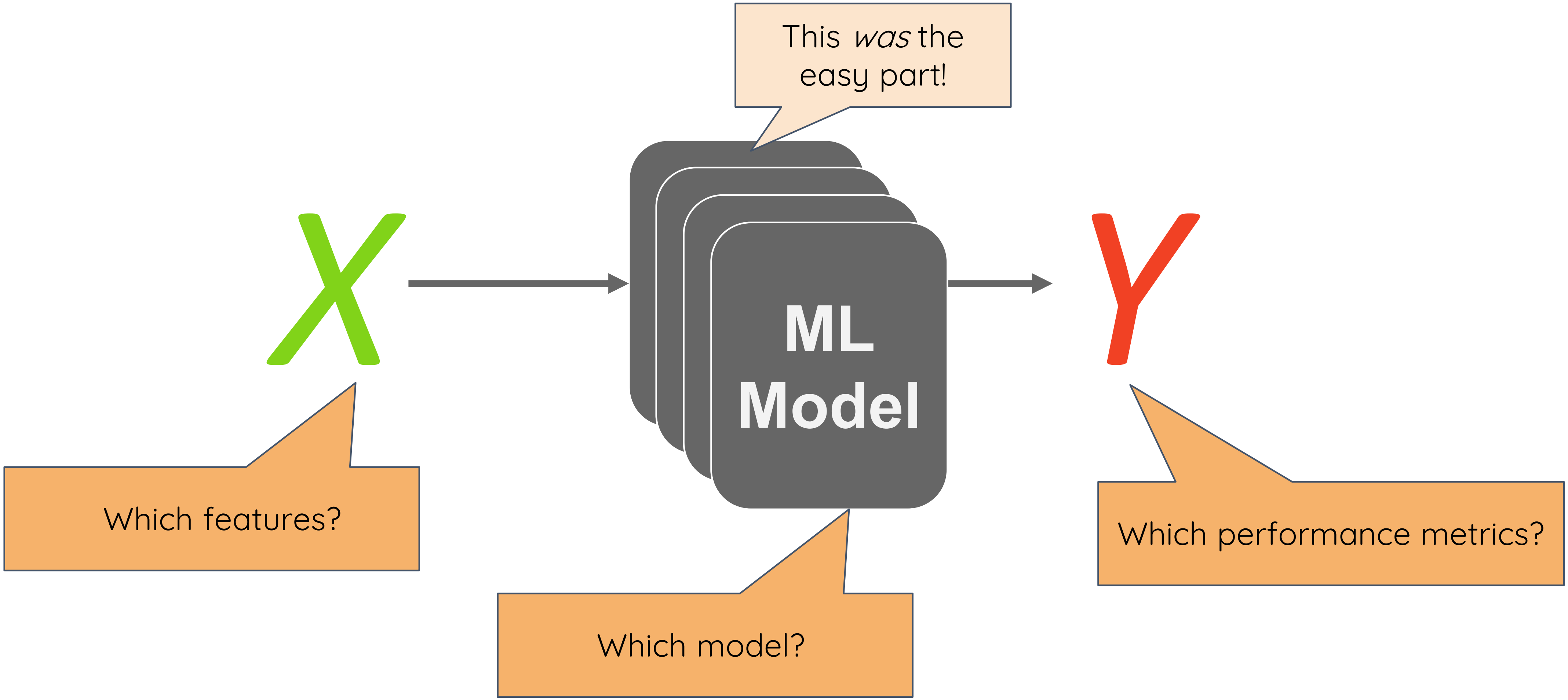
| Outcome | Supervised Learning | Unsupervised Learning |
|---|---|---|
| Continuous | Regression  | Principal Component  |
| Categorical | Classification  | Clustering  |

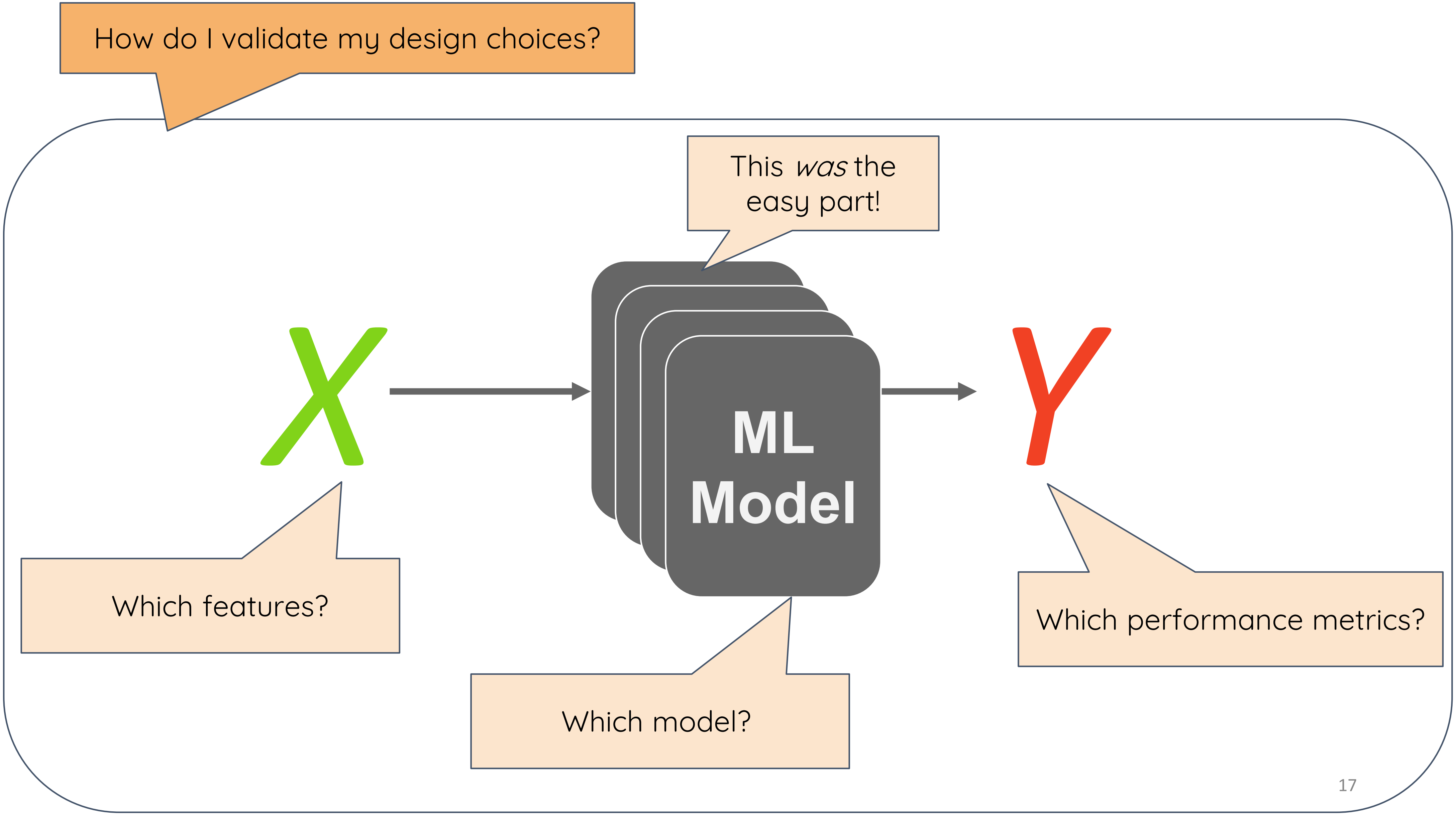
# Training a machine-learning model



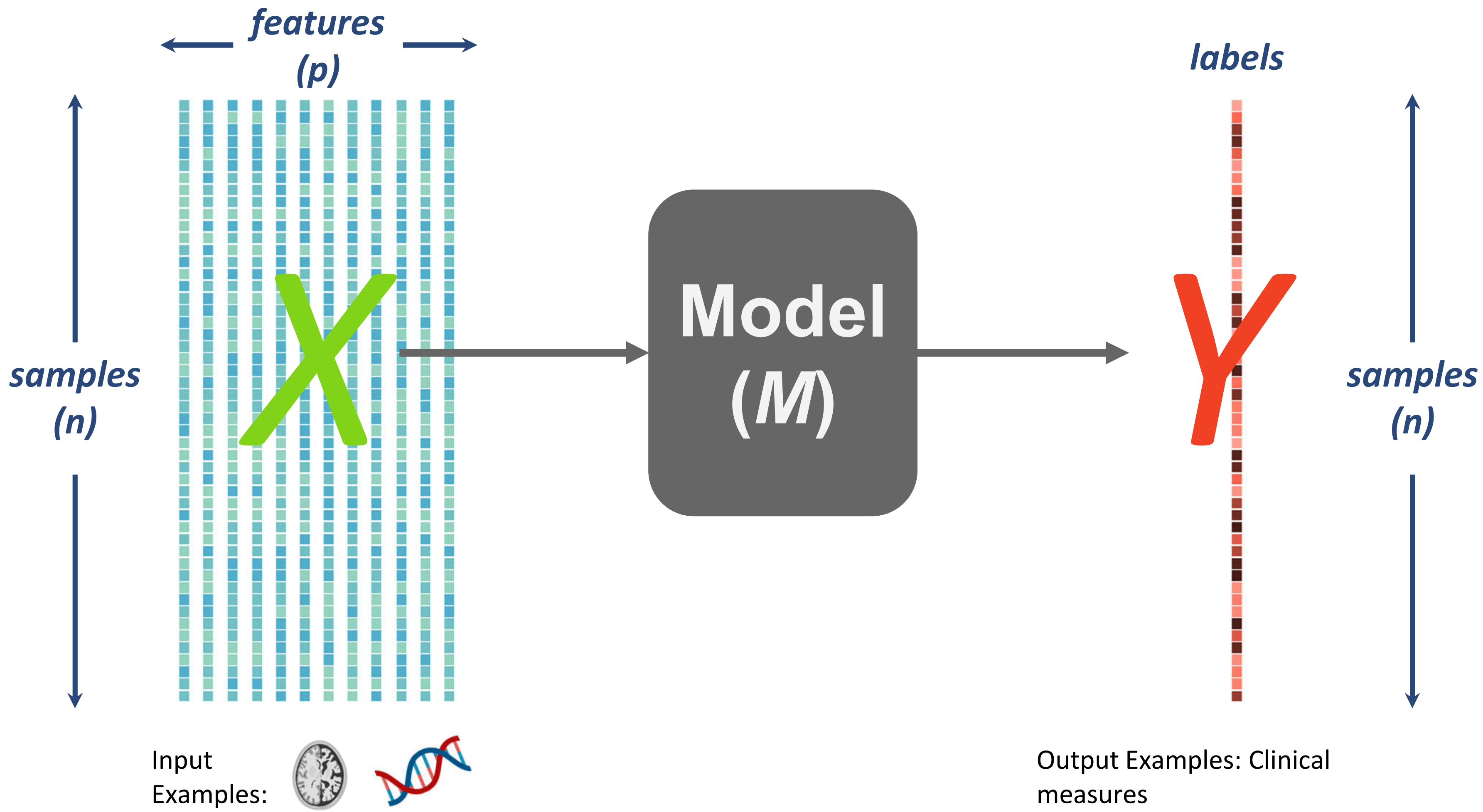
# Training a machine-learning model




This *was* the easy part!
X
ML Model
Y
Which features?
Which model?
Which performance metrics?


How do I validate my design choices?
This *was* the easy part!
X
ML Model
Y
Which features?
Which model?
Which performance metrics?

# Terminology

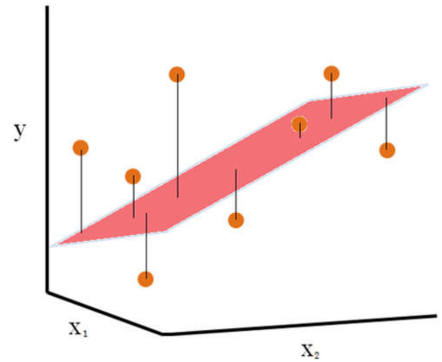


# Supervised Learning: Models

- Goal: *Learn* parameters (or weights) of a model (M) that maps **X** to **y**

# Supervised Learning: Models

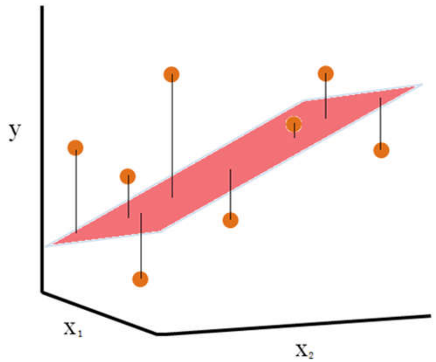
- Goal: *Learn* parameters (or weights) of a model (M) that maps X to y
- Example models:
  - Linear / Logistic regression



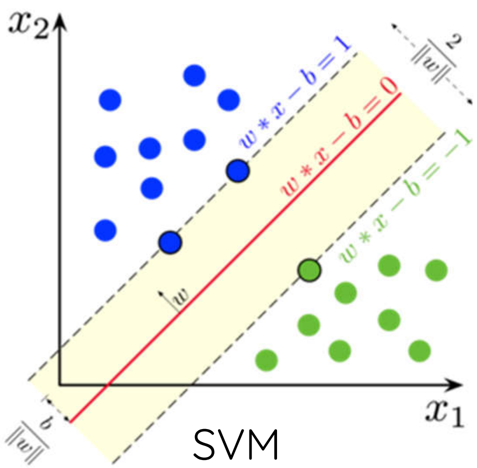

Linear Regression

# Supervised Learning: Models

- Goal: *Learn* parameters (or weights) of a model (M) that maps X to y
- Example models:
    - Linear / Logistic regression
    - Support vector machines



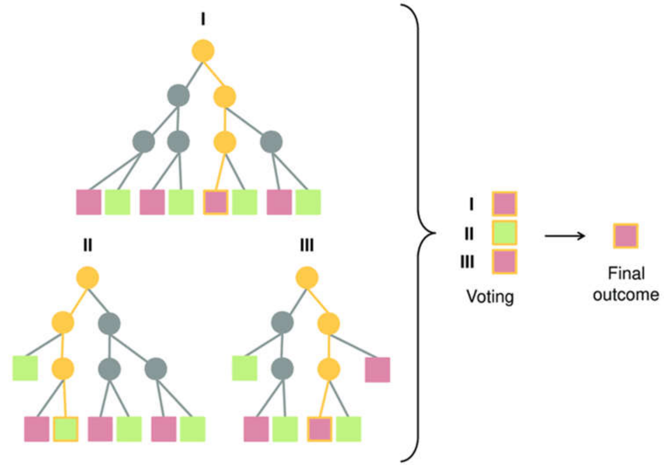

Linear Regression



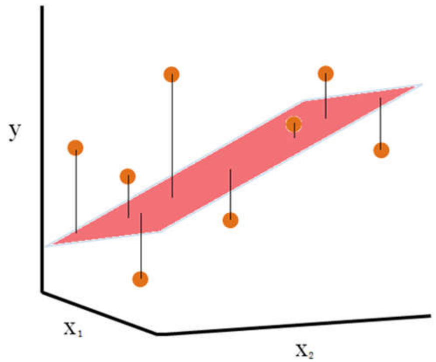

SVM

# Supervised Learning: Models

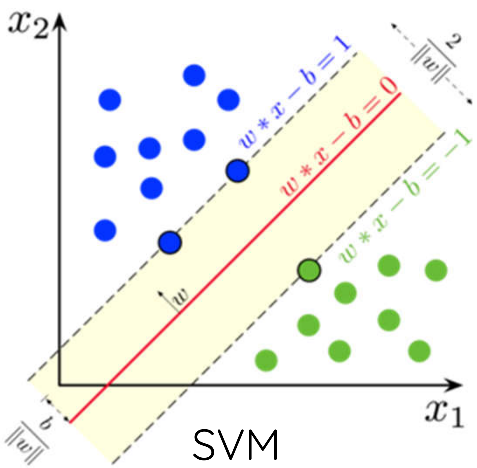
- Goal: *Learn* parameters (or weights) of a model (M) that maps X to y
- Example models:
  - Linear / Logistic regression
  - Support vector machines
  - Tree-ensembles: random forests, gradient boosting




Tree-ensembles



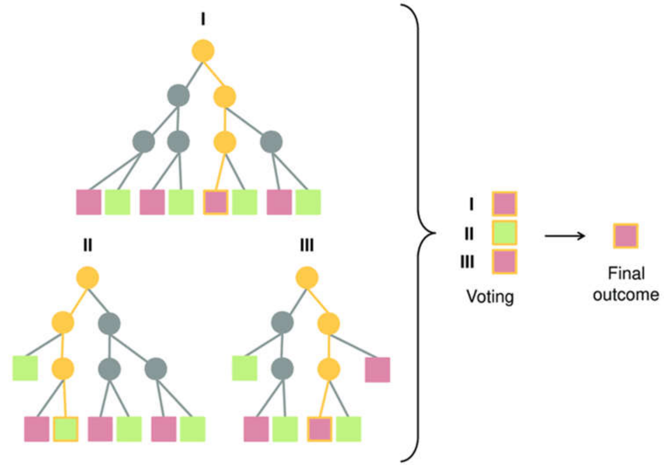

Linear Regression



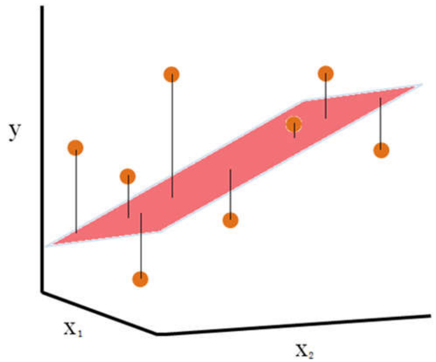

SVM

# Supervised Learning: Models

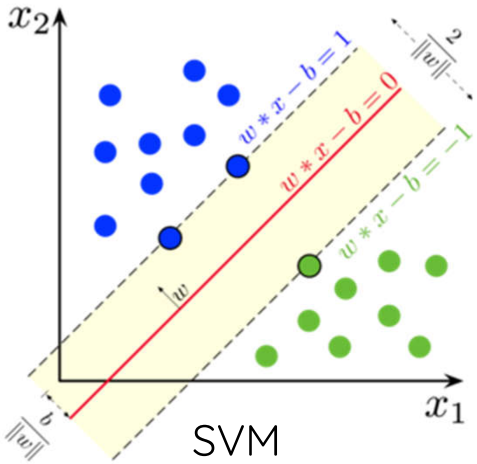
- Goal: *Learn* parameters (or weights) of a model (M) that maps X to y
- Example models:
  - Linear / Logistic regression
  - Support vector machines
  - Tree-ensembles: random forests, gradient boosting
  - Artificial Neural networks



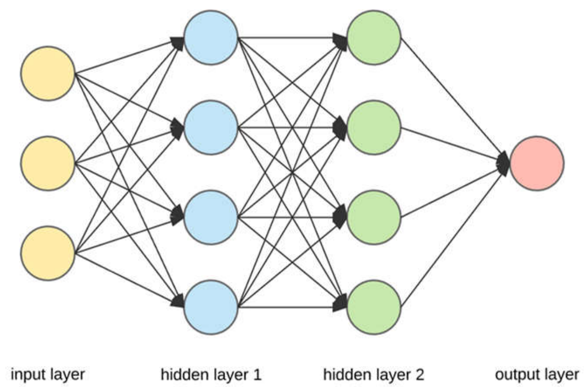

Tree-ensembles




Linear Regression



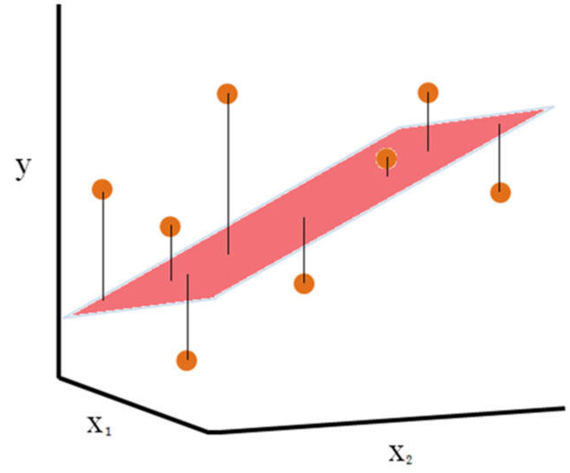

SVM

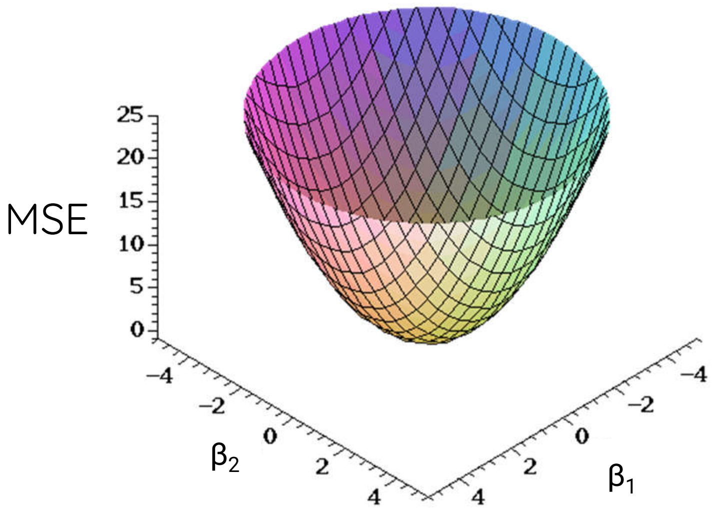



ANN

# Model Fitting

- How do we learn the model weights?
  - Example: Linear regression
  - Model: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$
  - Loss function: $\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
  - Optimization: Gradient descent



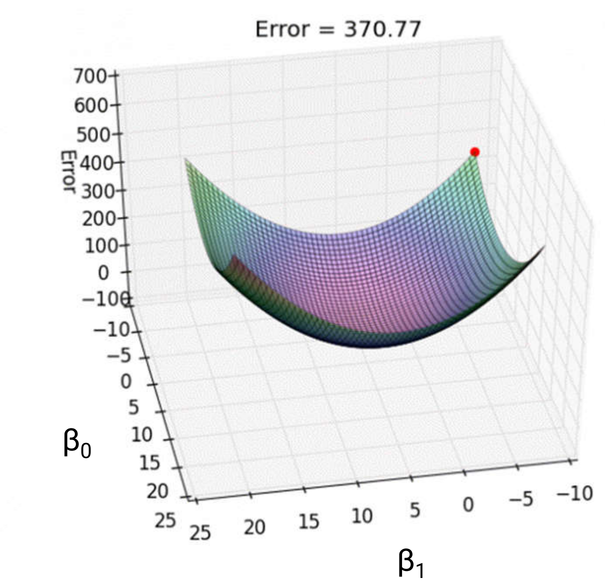



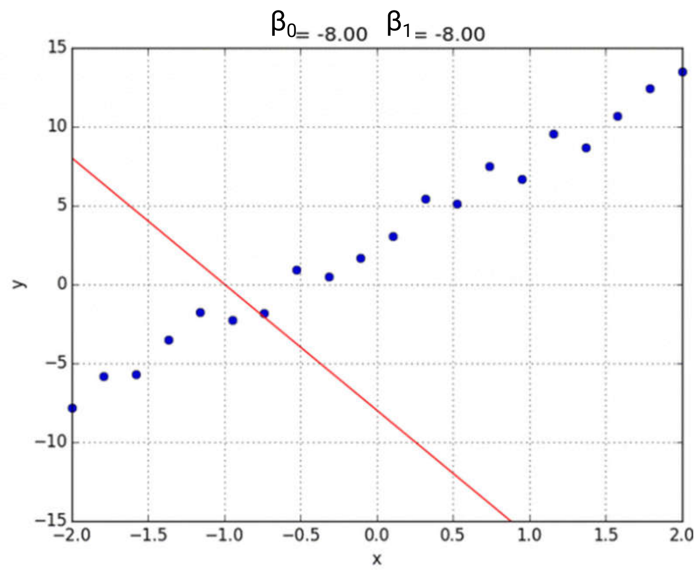
# Model Fitting

- Gradient descent with a **single** input variable and **n** samples
  - Start with random weights ($\beta_0$ and $\beta_1$)
  - Compute loss (i.e. MSE)
  - Update weights based on the gradient

$$\hat{y}_i = \beta_0 + \beta_1 x_i$$

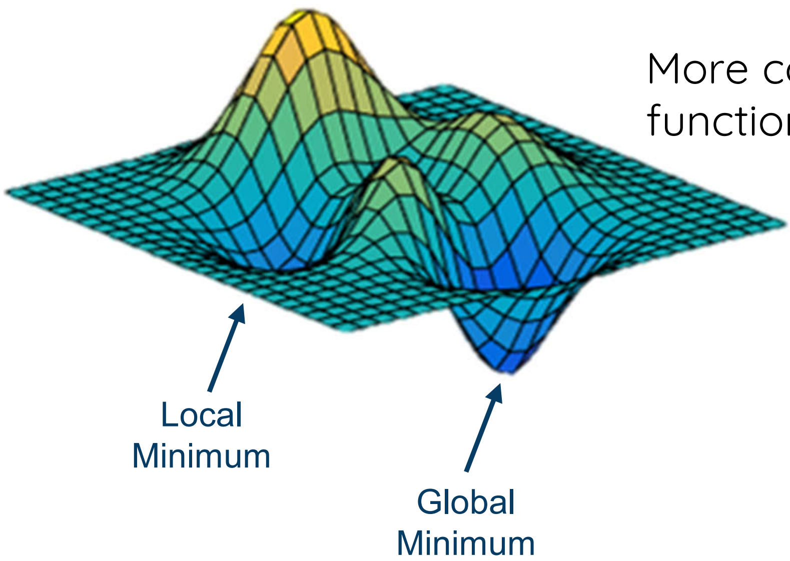
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$



# Model Fitting

- Gradient descent for complex models with non-convex loss functions
  - Start with random weights ($\beta_0$ and $\beta_1$)
  - Compute loss
  - Update weights based on the gradient




More complex models / loss functions (e.g. ANNs)

# Model Fitting

- Can we control this fitting process to get a model with specific characteristics?

# Model Fitting

- Can we control this fitting process to get a model with specific characteristics?
  - We have strong prior beliefs about what is a **plausible** model
    - e.g. I believe a disease symptom can be predicted with few genes.
  - Practical reasons
    - Prevent overfitting (n_features >> n_samples)

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_{p-1} x_{p-1} + \beta_p x_p$$

# Model Fitting

- Can we control this fitting process to get a model with specific characteristics?
  - We have strong prior beliefs about what is a **plausible** model
    - e.g. I believe a disease symptom can be predicted with few genes.
  - Practical reasons
    - **Prevent overfitting (n_features >> n_samples)**

- **Yes! → Model regularization**

# Model Fitting: Regularization

- How do we do it?
  - Modify the loss function
  - Constrain the learning process

- Examples:
  - L1 i.e. Lasso
  - L2 i.e. Ridge

# Model Fitting: Regularization

- How do we do it?
  - Modify the loss function
  - Constrain the learning process

- Examples:
  - **L1 i.e. Lasso**
  - **L2 i.e. Ridge**

1) L1/Lasso: constrains parameters to be ***sparse***

$$MSE = \sum_{i=1}^{n} (y_i - \underbrace{[\, \beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j \,]}_{\hat{y}_i})^2 + \underbrace{\lambda \sum_{j=1}^{p} |\beta_j|}_{L_1}$$

1) L2/Ridge: constrains parameters to be ***small***

$$MSE = \sum_{i=1}^{n} (y_i - \underbrace{[\, \beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j \,]}_{\hat{y}_i})^2 + \underbrace{\lambda \sum_{j=1}^{p} \beta_j^2}_{L_2}$$

# Model Fitting: Scikit-learn syntax

```
# import
from sklearn import linear_model, svm


# data
X = [[0, 0], [1, 1]]
y = [0, 1]
```

# Model Fitting: Scikit-learn syntax

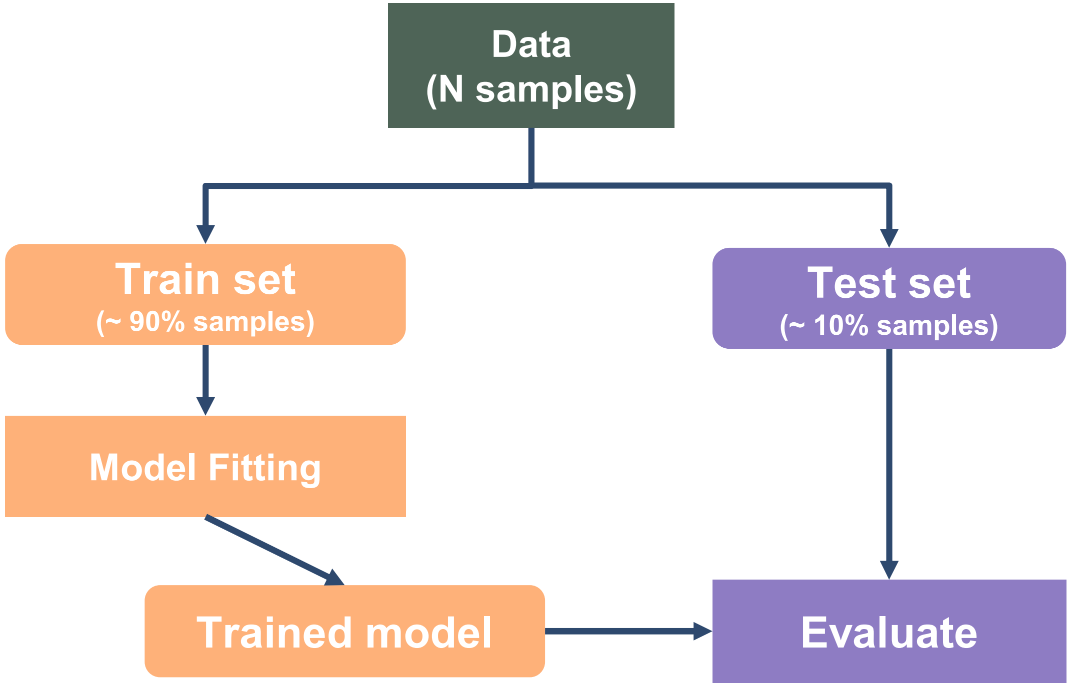
```
# import
from sklearn import linear_model, svm

# data
X = [[0, 0], [1, 1]]
y = [0, 1]

# pick a model
model = linear_model.Lasso(alpha=0.1)   # model = svm.SVC()

# fit the model with data
model.fit(X, y)
```

# Model Fitting: Scikit-learn syntax

```
# import
from sklearn import linear_model, svm

# data
X = [[0, 0], [1, 1]]
y = [0, 1]

# pick a model
model = linear_model.Lasso(alpha=0.1)   # model = svm.SVC()

# fit the model with data
model.fit(X, y)

# predict on new data
y_pred = model.predict([[1, 0]])
```

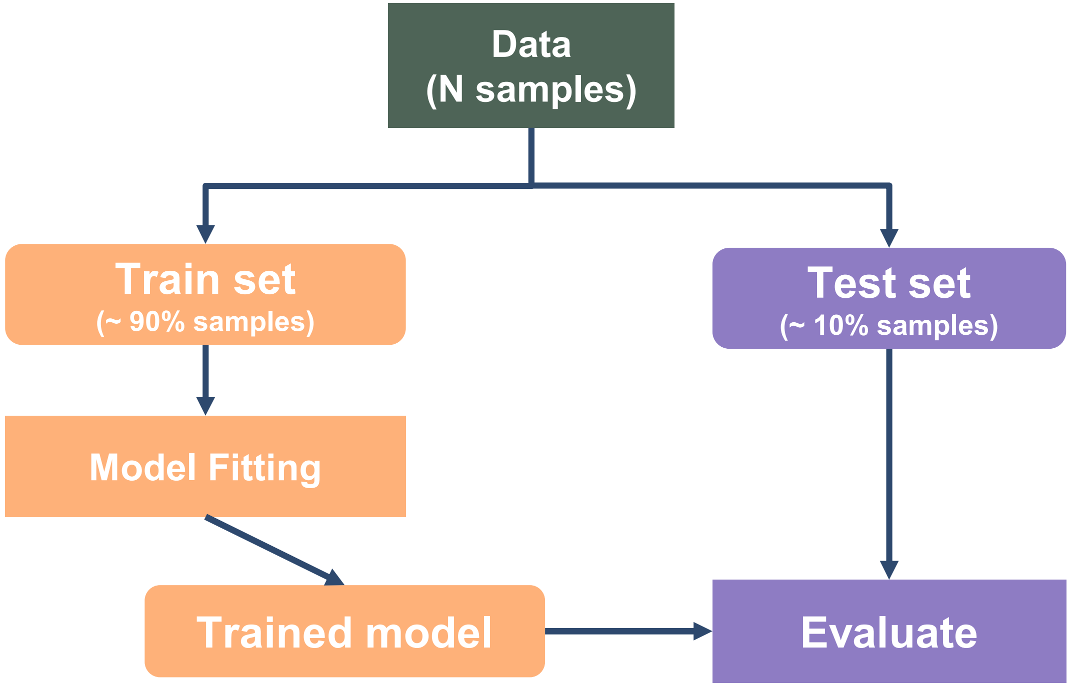
# Model Evaluation

- Is the model generalizable?
- How do we sample train and test sets?
- How do we select a model?



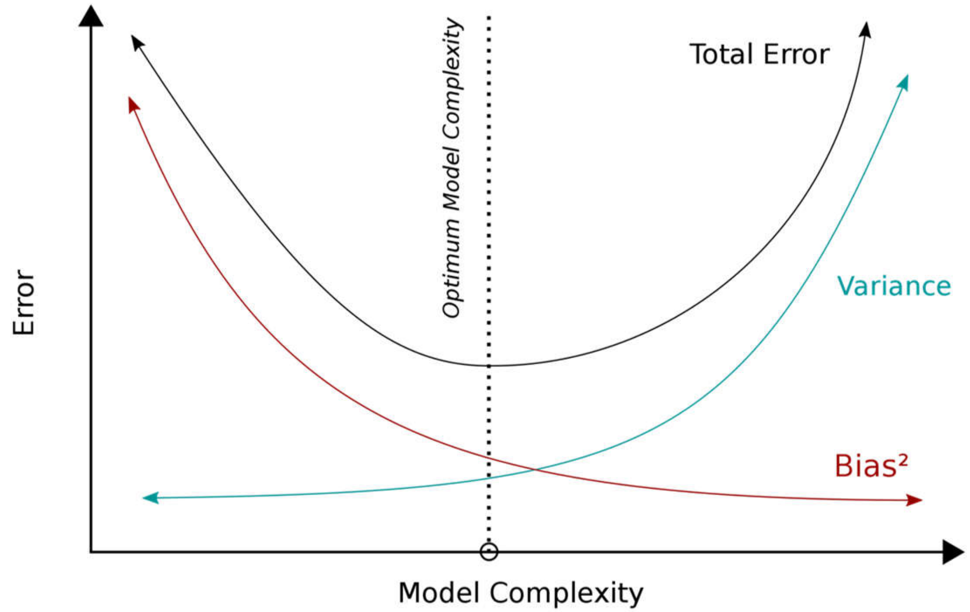
# Model Evaluation

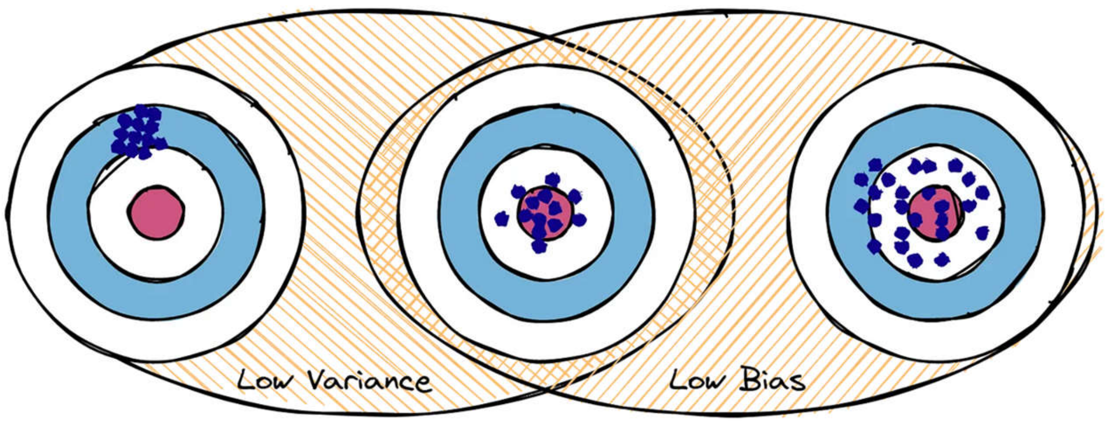
- **Is the model generalizable?**
- How do we sample train and test sets?
- How do we select a model?



# Model Evaluation

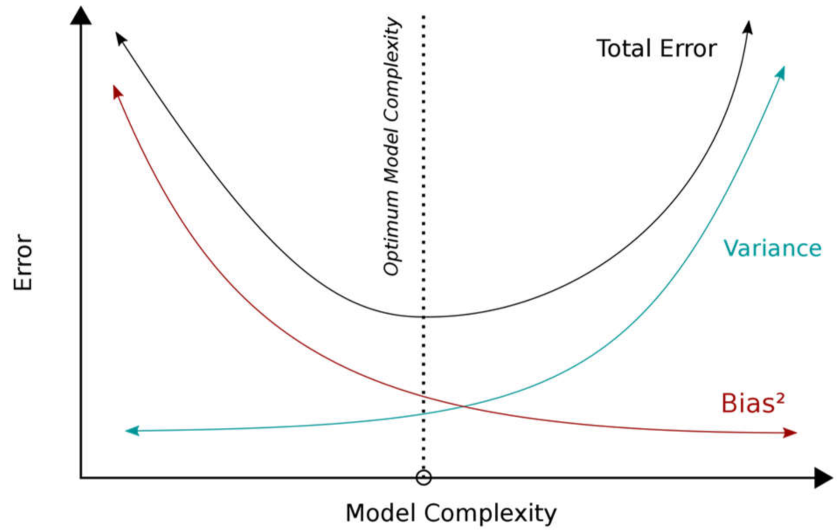
- Train performance ≠ Test performance
  - Model: Underfitting vs Overfitting
  - Errors: Bias - Variance tradeoff

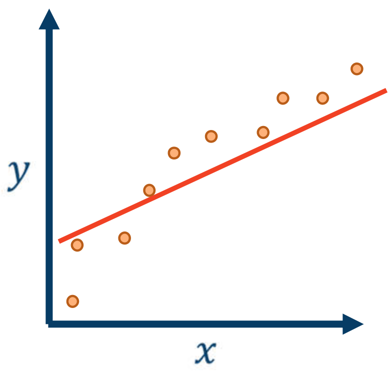


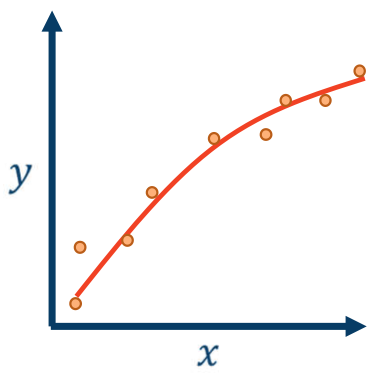
# Model Evaluation

- Train performance ≠ Test performance
  - Model: Underfitting vs Overfitting
  - Errors: Bias - Variance tradeoff
  - Regression example

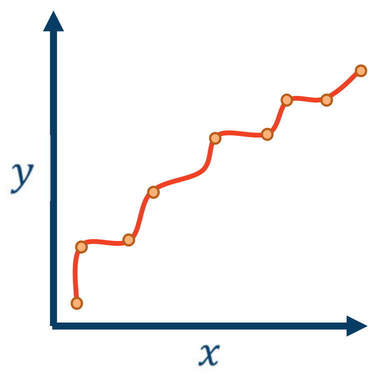



Underfitting




Optimal

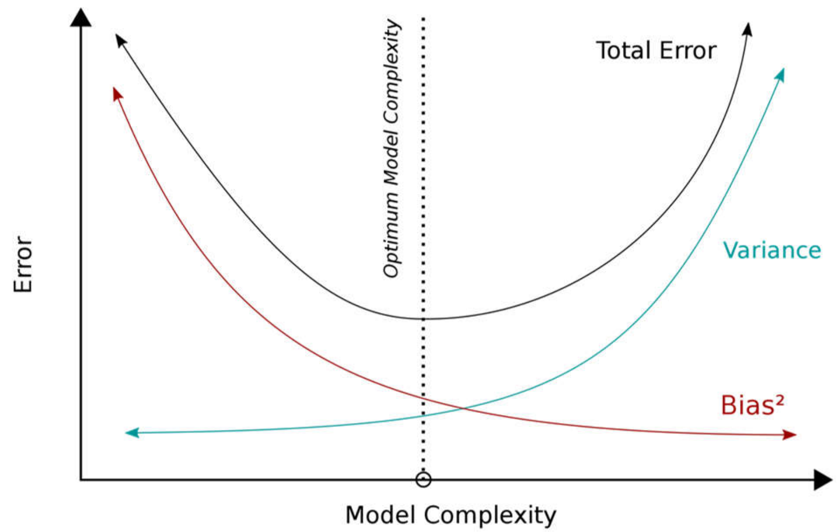


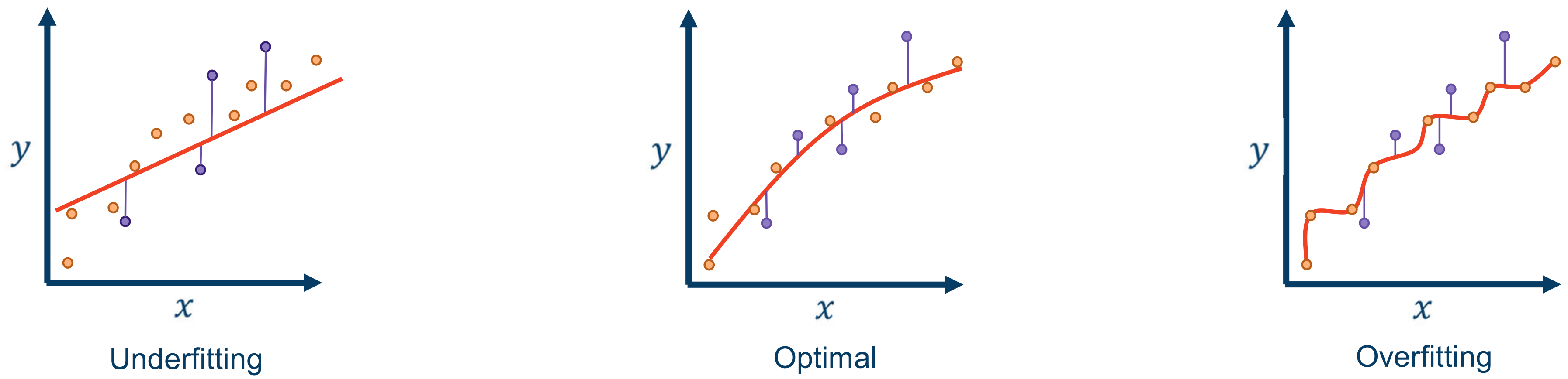

Overfitting

Train set

# Model Evaluation

- Train performance ≠ Test performance
  - Model: Underfitting vs Overfitting
  - Errors: Bias - Variance tradeoff
  - Regression example

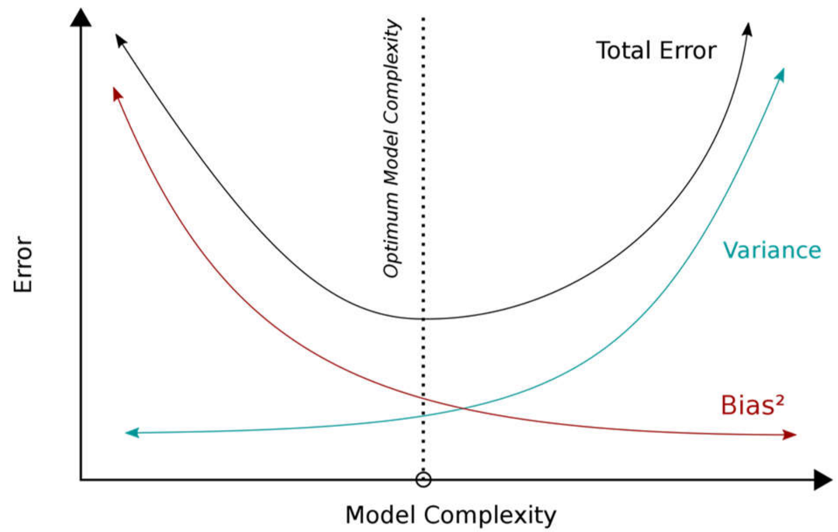


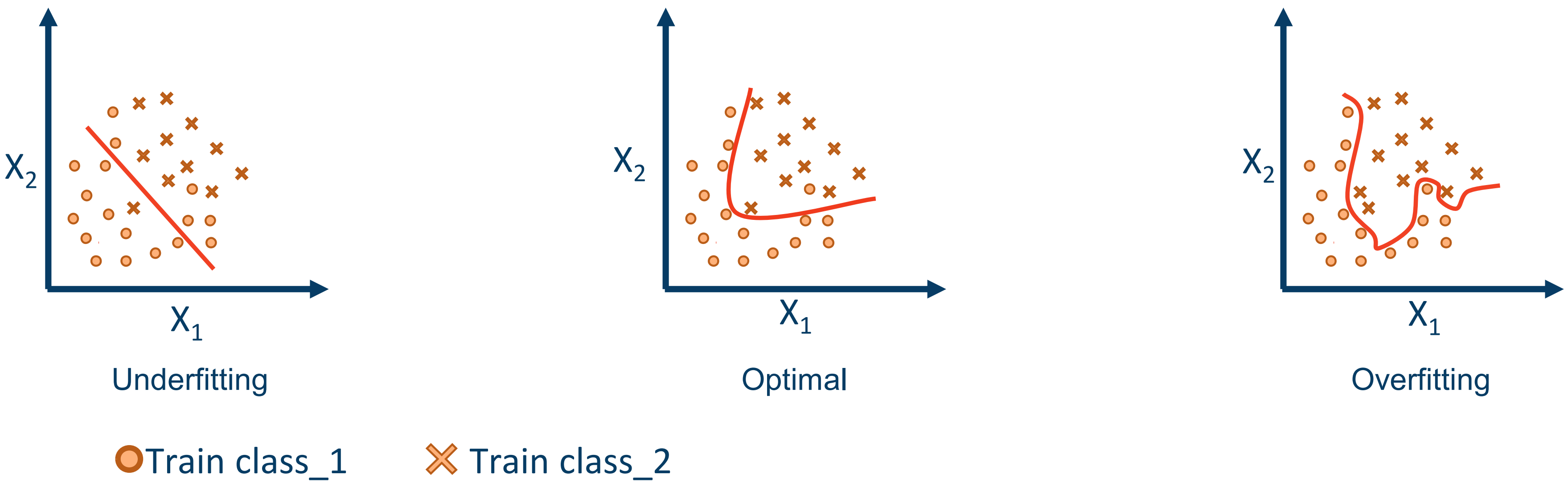




Train set Test set

# Model Evaluation

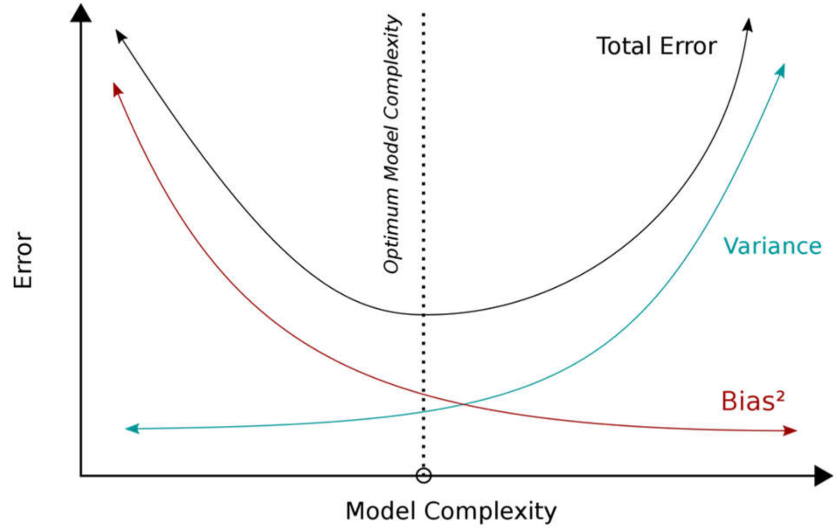
- Train performance ≠ Test performance
  - Model: Underfitting vs Overfitting
  - Errors: Bias - Variance tradeoff
  - Classification example (i.e. separate **"o"** vs **"x"**)

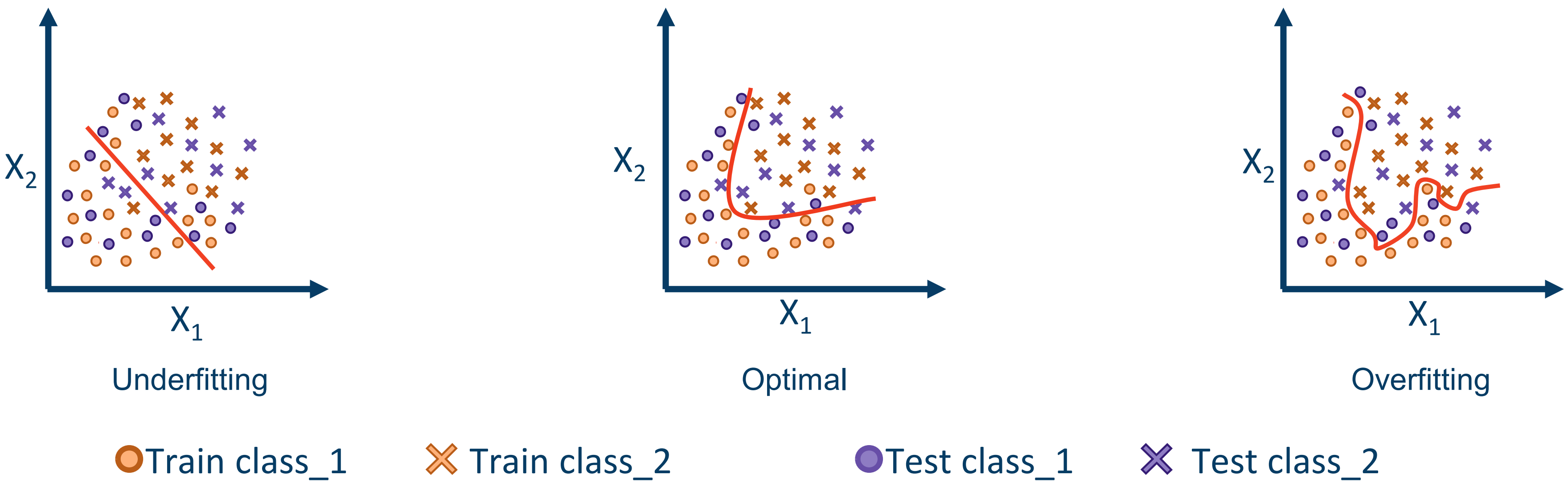


# Model Evaluation

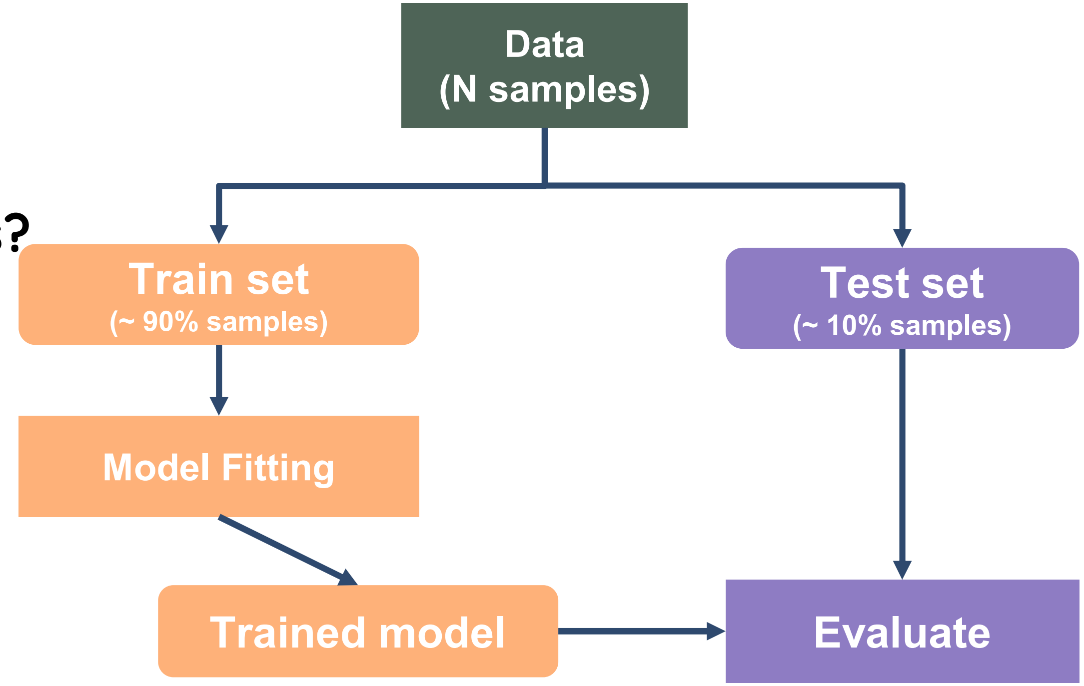
- Train performance ≠ Test performance
  - Model: Underfitting vs Overfitting
  - Errors: Bias - Variance tradeoff
  - Classification example (i.e. separate **“o”** vs **“x”**)



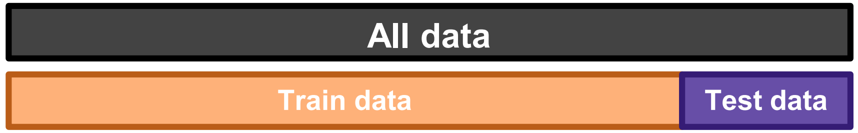
# Model Evaluation

- Is the model generalizable?
- **How do we sample train and test sets?**
- How do we select a model?



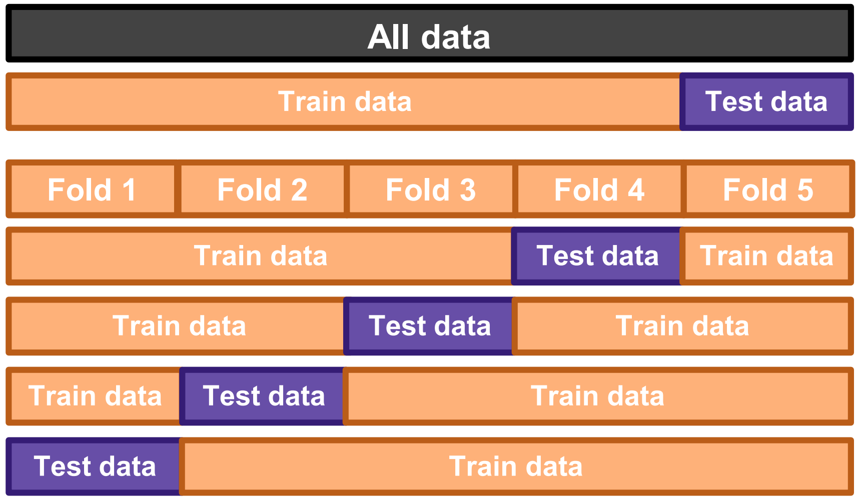
# Model Evaluation: Cross-Validation (Outer loop)

- How do we sample train and test sets?
  - Train set: learn model parameters
  - Test set (a.k.a held-out sample): Evaluate model performance



# Model Evaluation: Cross-Validation (Outer loop)

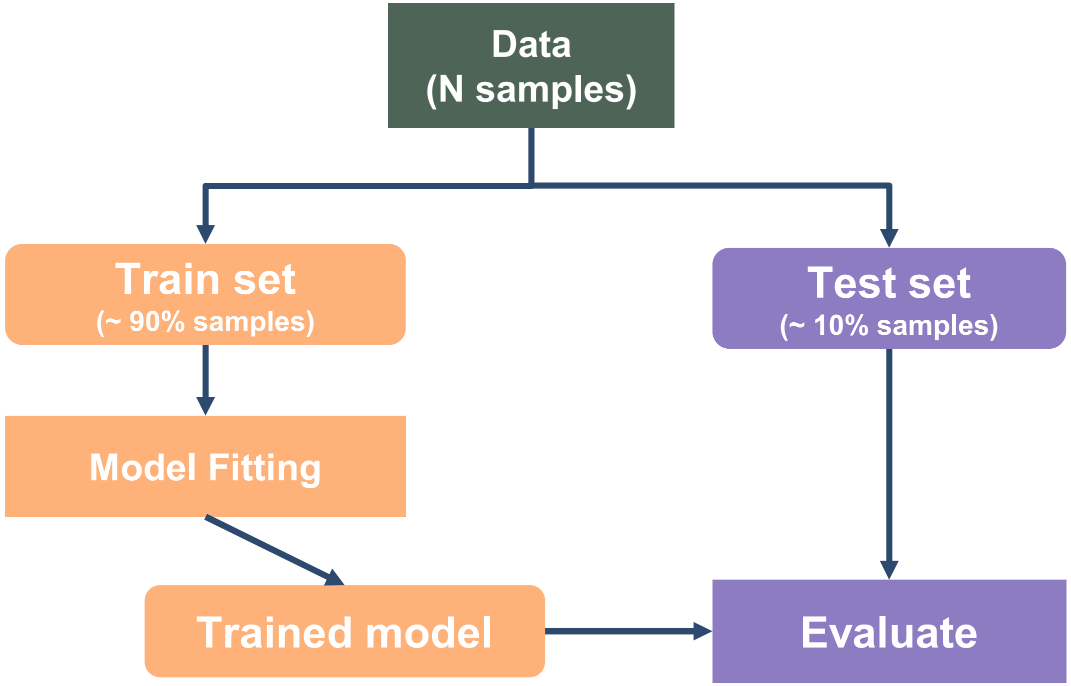
- How do we sample train and test sets?
  - Train set: learn model parameters
  - Test set (a.k.a held-out sample): Evaluate model performance
  - Repeat for different Train-Test splits
    - k-fold, shuffle-split
  - Report performance statistics over all test folds




CV outer loop

# Model Evaluation

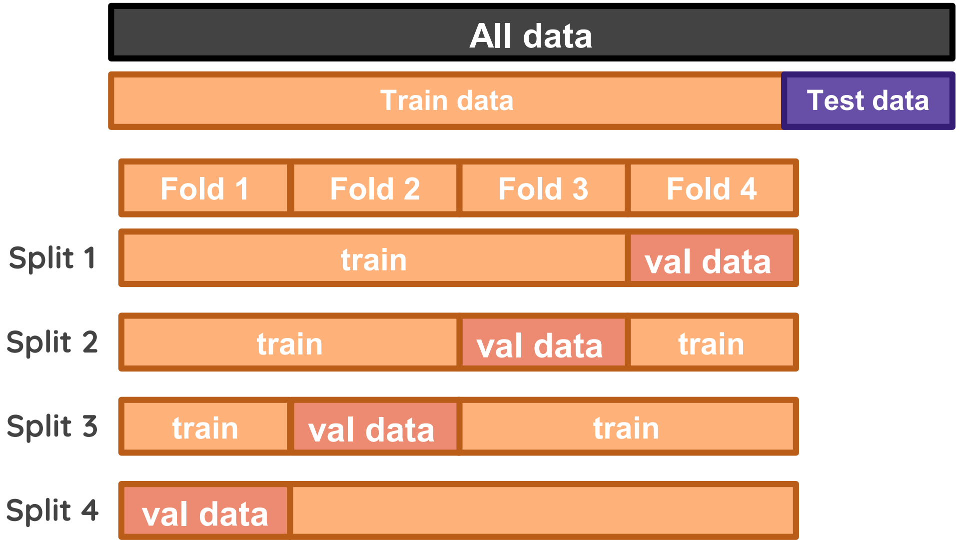
- Is the model generalizable?
- How do we sample train and test sets?
- **How do we select a model?**



# Model Evaluation: Cross-Validation (Inner loop)

- How do we select a model?
  - Tune *hyper-parameters* of a model
  - Compare several different model architectures
  - Select / transform raw features

- This repeats for all train-test splits in the outer loop




CV inner loop

# Model Evaluation: Hyper-parameters

- Hyper-parameter ≠ parameter (or weights)
  - Parameters are **learned**; hyper-parameters are **chosen**!

# Model Evaluation: Hyper-parameters

- Hyper-parameter ≠ parameter (or weights)
  - Parameters are **learned**; hyper-parameters are **chosen**!

- Examples:
  - Degree of model (eg. linear vs quadratic)
  - Kernels
  - Number of trees
  - Number of layers, filters, batch-size, learning-rate in ANNs

# Model Evaluation: Hyper-parameters

- Hyper-parameter ≠ parameter (or weights)
  - Parameters are **learned**; hyper-parameters are **chosen**!

- Examples:
  - Degree of model (eg. linear vs quadratic)
  - Kernels
  - Number of trees
  - Number of layers, filters, batch-size, learning-rate in ANNs

- How do we choose them?
  - Prior beliefs → eg. cortical thickness and age have quadratic relationship.
  - Arbitrarily → we gotta start with something!
  - Trial and error → do a computationally feasible grid-search.

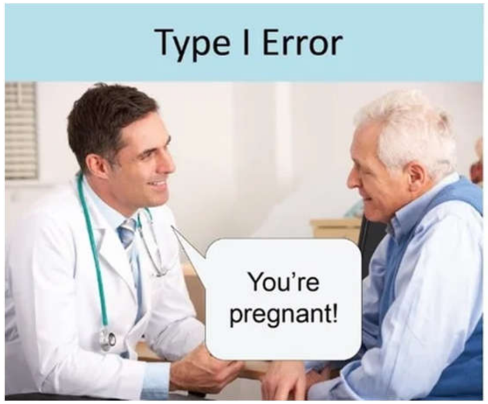
# Performance Scores

- Loss functions → computationally well-suited metrics
  - May / need not completely capture performance metrics of interest

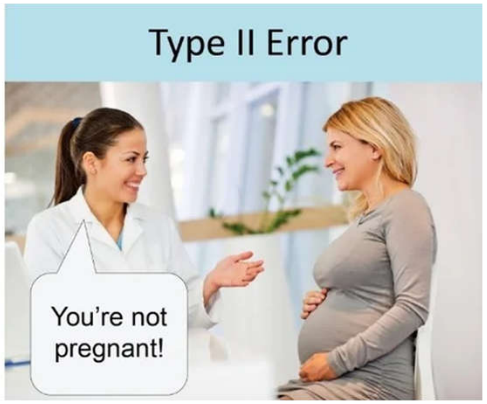
- Scores → practically useful metrics
  - Binary classification

| Confusion Matrix | | Ground Truth | |
|---|---|---|---|
| | | POSITIVE | NEGATIVE |
| Prediction | POSITIVE | TP | FP |
| | NEGATIVE | FN | TN |

False Positive




False Negative



# Performance Scores

- ML model that detects Covid from chest CTs. Current Covid prevalence ~ 1%.
  - FP: model predicts *Covid* when person is *healthy*
  - FN: model predicts *healthy* when person has *Covid*
- What happens if we build model that predicts everyone as healthy?
  - i.e. zero FPs!

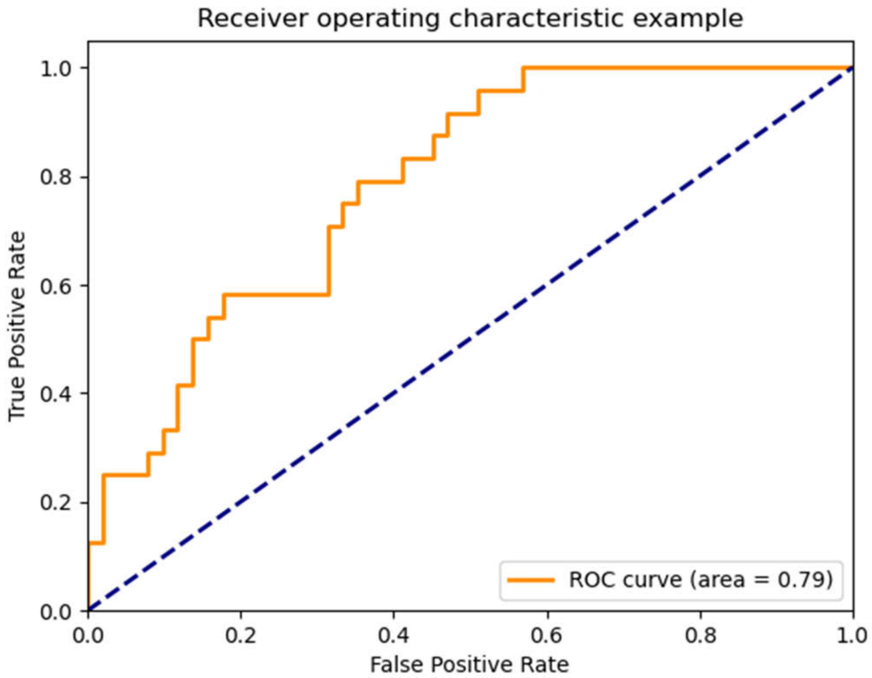
# Performance Scores

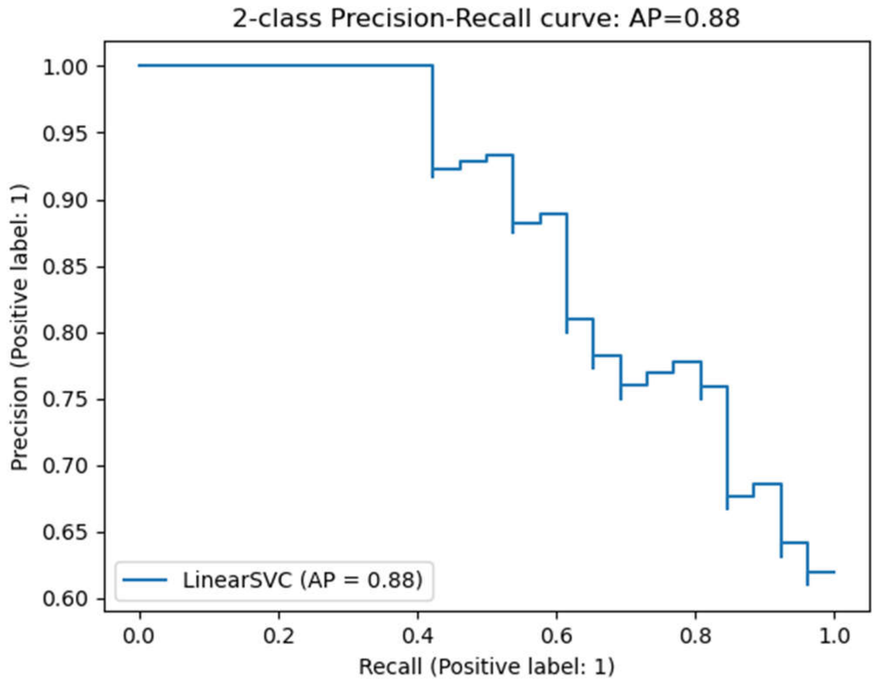
- ML model that detects Covid from chest CTs. Current Covid prevalence ~ 1%.
  - FP: model predicts *Covid* when person is *healthy*
  - FN: model predicts *healthy* when person has *Covid*
- What happens if we build model that predicts everyone as healthy?

| **Score** | **Formula** | **Null** | **What does it tell us?** | **When do I use it?** |
|---|---|---|---|---|
| **Accuracy** | (TP+TN) / (TP+FP+FN+TN) | 0.99 | How many people did we correctly predict out of all the people scanned? | FNs & FPs have similar costs |
| **Precision (i.e. PPV)** | TP/(TP+FP) | NaN | How many of those who we predicted as "covid" do actually have "covid"? | If you want to be more confident of your TPs |
| **Recall (aka Sensitivity)** | TP/(TP+FN) | 0 | Of all the people who have covid, how many of those did we correctly predict? | If you prefer FPs over FNs. |
| **Specificity** | TN/(TN+FP) | 1 | Of all the people who are healthy, how many of those did we correctly predict? | If you prefer FNs over FPs. |
| **F1** | 2*(Recall * Precision) / (Recall + Precision) | NaN | Harmonic mean(average) of the precision and recall. | When you have an uneven class distribution |

# Performance Curves

- Receiver Operating Characteristic (ROC) → Want high area-under-the-curve (AUC)
- Precision-Recall → Want high AUC or high Average precision (AP)



# Practical intuition

**Task: Segmentation, diagnosis etc**

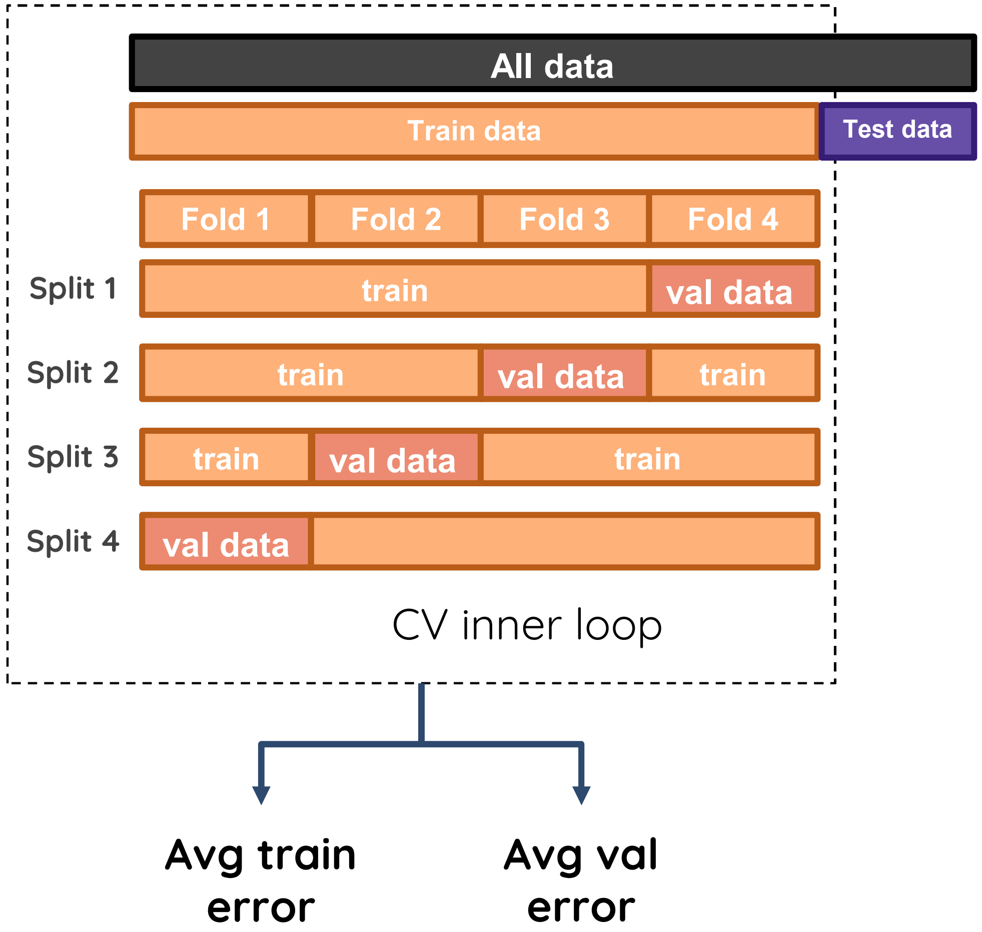
- Human error ~ 2%
- Train error ~ 10%
- Val error ~ 20%

Bias / underfit

Variance / overfit

**What do we do?**

- Underfitting → Bigger/different model
- Overfitting → More data / regularization



# Practical intuition

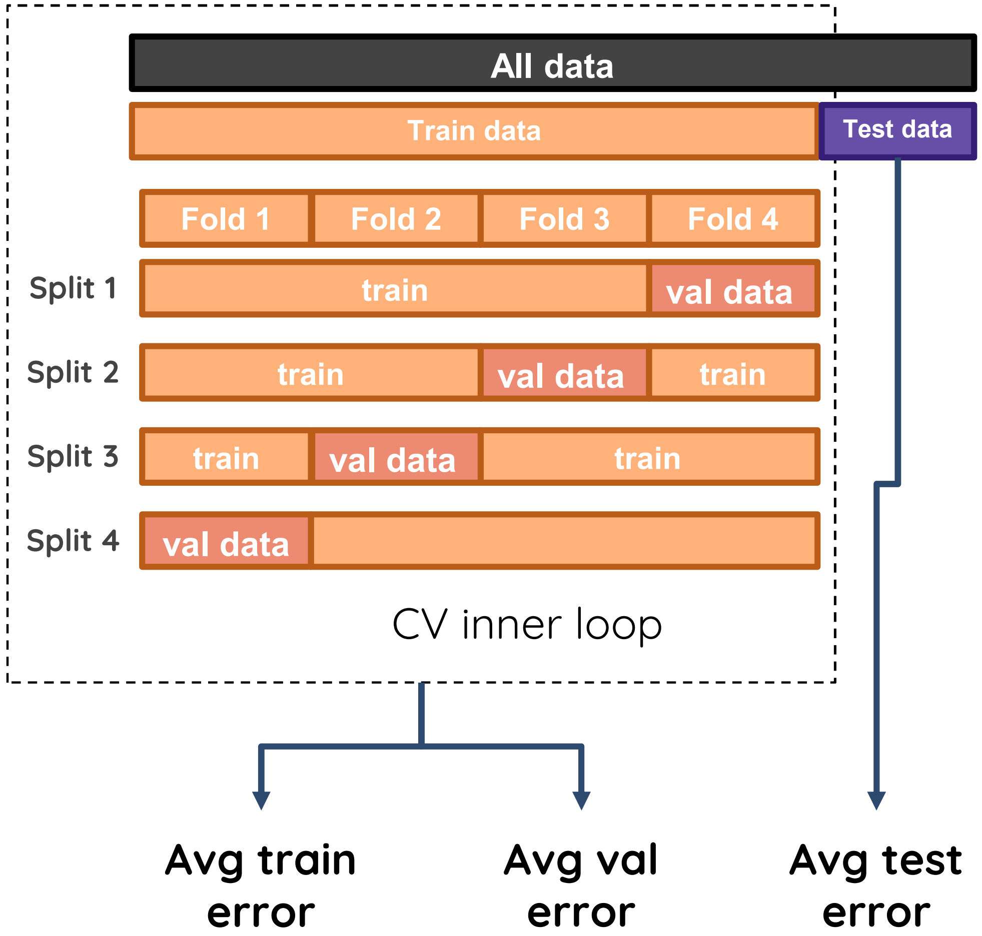
**Task: Segmentation, diagnosis etc**

- Human error ~ 2%
- Train error ~ 5%
- Val error ~ 5%
- Test error ~ 20%

dataset shift (overfit)

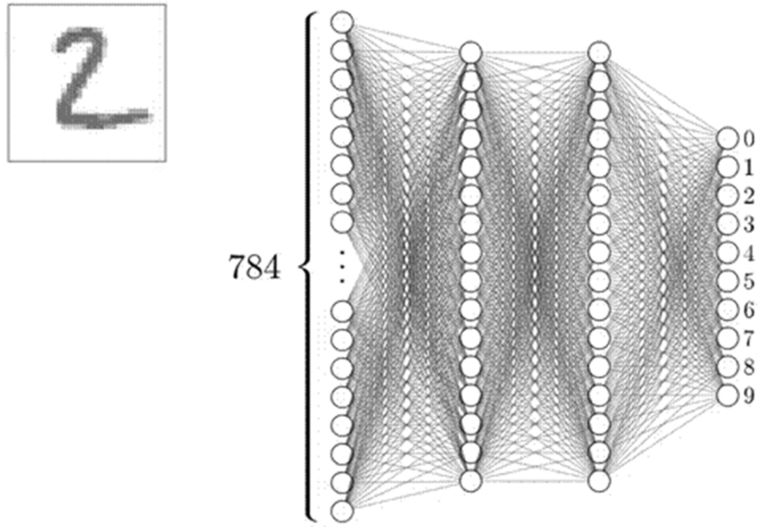
**What do we do?**

- Feature shift → get / generate more data
- Concept drift → fix / refine labels



# Deep-learning

- Why the buzz?
  - Works amazing on spatio-temporal input
  - Highly flexible → universal function approximator




ANN for handwritten-digit images
(gif source: 3b1b)

# Deep-learning

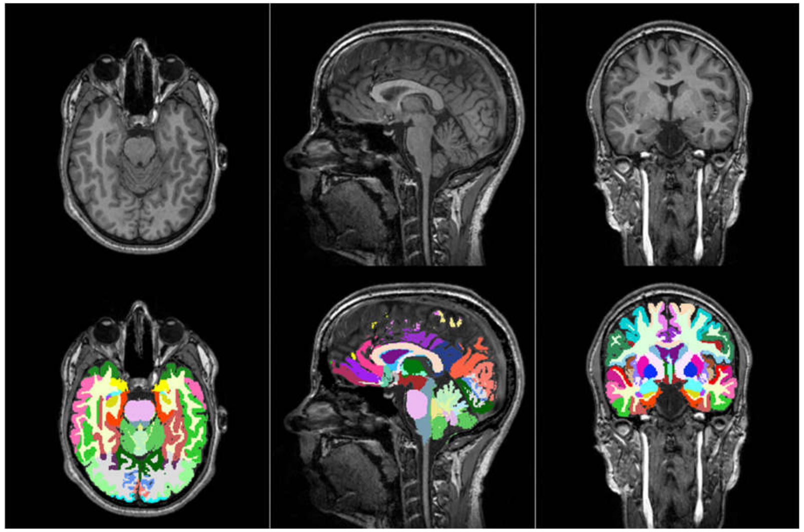
- Why the buzz?
  - Works amazing on spatio-temporal input
  - Highly flexible → universal function approximator

- What are the challenges?
  - Large number of parameters (175B!) → data hungry
  - Large number of hyper-parameters → difficult to train


LLM Transformers
(gif source: [3b1b])

# Deep-learning

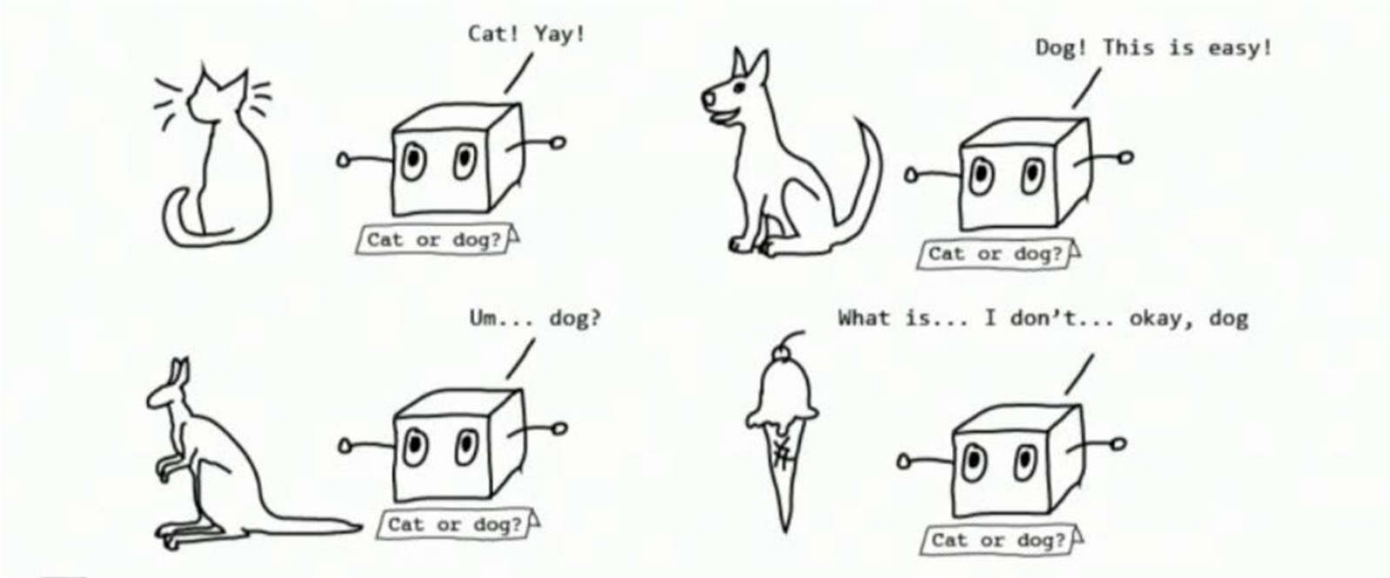
- Why the buzz?
  - Works amazing on spatio-temporal input
  - Highly flexible → universal function approximator
- What are the challenges?
  - Large number of parameters (175B!)→ data hungry
  - Large number of hyper-parameters → difficult to train
- When do I use it?
  - If you have highly-structured input, eg. medical images.
  - You have a lot of data and computational resources.



Source:
https://github.com/fepegar/torchio

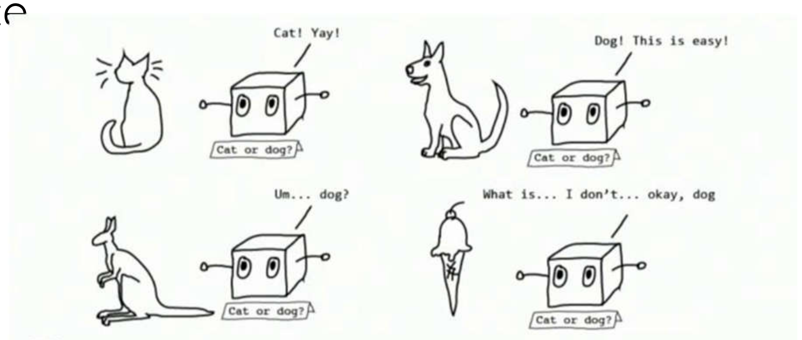
# Pitfalls and Challenges

- Models do not generalize even after good CV performance
  - Implicit double-dipping
  - Dataset biases (eg. North-American demographics)
  - Noisy labels (eg. diagnosis definitions)
  - Data distribution shifts (eg. assay, scanner upgrades)



# Pitfalls and Challenges

- Models do not generalize even after good CV performance
  - Implicit double-dipping
  - Dataset biases (eg. North-American demographics)
  - Noisy labels (eg. diagnosis definitions)
  - Data distribution shifts (eg. assay, scanner upgrades)




- Unnecessary complexity
  - Do I really need a giant deep-net or a simple linear model would do?

# ML Novice Checklist

- Data
  - What is my n_features and n_samples?
  - Am I encoding categorical data correctly?
  - Am I using information (e.g. mean) from test set to preprocess (eg. zscore) the data?

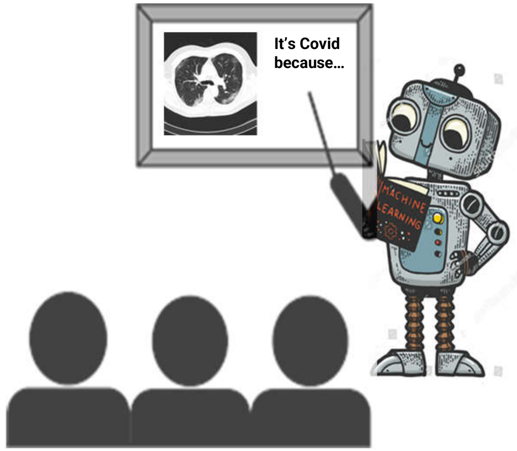
# ML Novice Checklist

- Data
  - What is my n_features and n_samples?
  - Am I encoding categorical data correctly?
  - Am I using information (e.g. mean) from test set to preprocess (eg. zscore) the data?

- Model
  - Do my performance metrics capture the practical use-case of interest?
  - What is the null / dummy model performance?
    - Classification: Predict majority class all the time
    - Regression: Predict the median value all the time
  - Am I interpreting model parameters (i.e. weights) correctly?

# Takeaways

- Supervised ML is useful for **predictions** but **not really for explanations**
  - eg. image segmentation, prognosis, drug development
- Our job is to ensure **generalizability** of these models
  - Multitude of validations
  - Understanding model biases and limitations
- ***Engineering tools*** vs *Scientific discovery*
  - Interpretability and explainability




Explainable AI

# Food for thought

- Ethical dilemmas
- Socialietal implications
- What's real?

***Ceci n'est pas un pape***

# ML for Neuroimaging

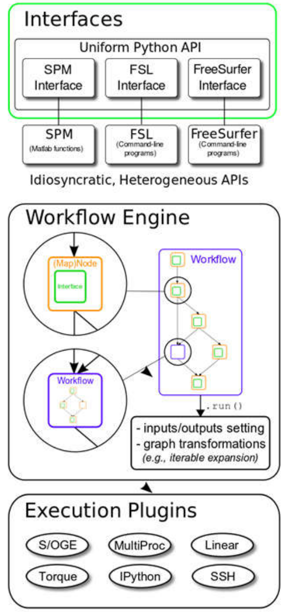
- Why Neuroimaging is special?
  - Imaging data is huge (Dimention reduction)
  - Imaging data has multi-modality (Fusion)
  - Imaging data is noisy (indirect measure of brain activities)

# Python tools for Neuroimaging ML

- **Community of practice**: Nipy.org
- Pipelines and interfaces: Nipype
- Anatomy: dipy, Mindboggle
- File I/O and data management: nibabel, Scitran SDM, pybids…
- fMRI: Nipy, Nitime, popeye…
- ML: **nilearn**, PyMVPA…
- i/S/M/EEG: MNE…
- Visualization: napari-nibabel, niwidgets…

**Nipype: Neuroimaging in Python Pipelines and Interfaces**

ni learn

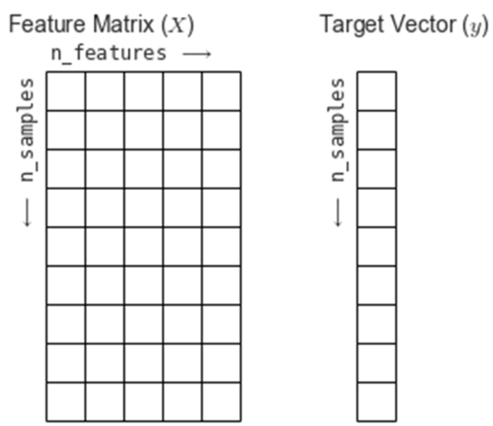



https://nipy.org/
https://nipype.readthedocs.io/en/latest/index.html
https://nilearn.github.io/dev/index.html

# Nilearn

Nilearn enables approachable and versatile analyses of brain volumes. It provides **statistical and machine-learning tools**, with instructive documentation & open community.

It supports general linear model (GLM) based analysis and leverages the scikit-learn Python toolbox for multivariate statistics with applications such as predictive modelling, classification, decoding, or connectivity analysis.

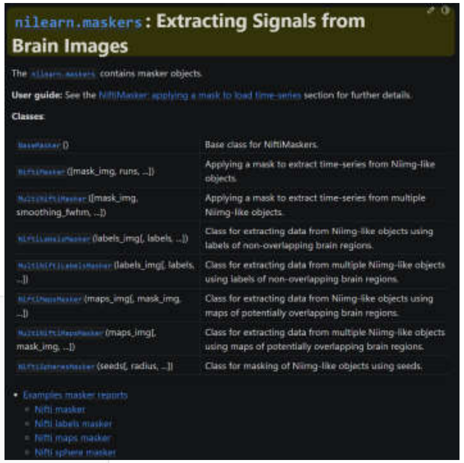


# Nilearn example: Extracting features from Imaging data

```python
from nilearn import datasets

development_dataset = datasets.fetch_development_fmri(n_subjects=30)
```

```python
import nibabel as nib

# Subset to just the first image
img = nib.load(development_dataset.func[0])
img.shape
```

```python
import numpy as np

msdl_atlas = datasets.fetch_atlas_msdl()

msdl_coords = msdl_atlas.region_coords
n_regions = len(msdl_coords)

print(f'MSDL has {n_regions} ROIs, part of the following networks :\n{np.unique(msdl_atlas.network
```



**nilearn.maskers: Extracting Signals from Brain Images**

The nilearn.maskers contains masker objects.

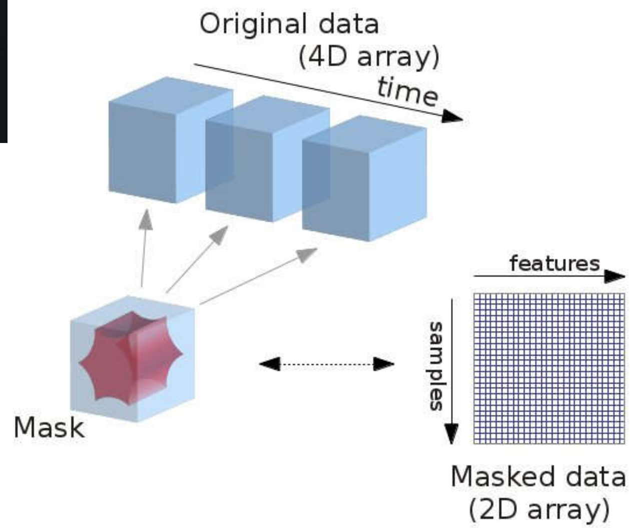
**User guide:** See the NiftiMasker: applying a mask to load time-series section for further details.

**Classes:**

| | |
|---|---|
| BaseMasker () | Base class for NiftiMaskers. |
| NiftiMasker ([mask_img, runs, ...]) | Applying a mask to extract time-series from Niimg-like objects. |
| MultiNiftiMasker ([mask_img, smoothing_fwhm, ...]) | Applying a mask to extract time-series from multiple Niimg-like objects. |
| NiftiLabelsMasker (labels_img[, labels, ...]) | Class for extracting data from Niimg-like objects using labels of non-overlapping brain regions. |
| MultiNiftiLabelsMasker (labels_img[, labels, ...]) | Class for extracting data from multiple Niimg-like objects using labels of non-overlapping brain regions. |
| NiftiMapsMasker (maps_img[, mask_img, ...]) | Class for extracting data from Niimg-like objects using maps of potentially overlapping brain regions. |
| MultiNiftiMapsMasker (maps_img[, mask_img, ...]) | Class for extracting data from multiple Niimg-like objects using maps of potentially overlapping brain regions. |
| NiftiSpheresMasker (seeds[, radius, ...]) | Class for masking of Niimg-like objects using seeds. |

- Examples masker reports
  - Nifti masker
  - Nifti labels masker
  - Nifti maps masker
  - Nifti sphere masker

```python
from nilearn import input_data

masker = input_data.NiftiMapsMasker(
    msdl_atlas.maps, resampling_target="data",
    t_r=2, detrend=True,
    low_pass=0.1, high_pass=0.01).fit()
```




```python
roi_time_series = masker.transform(development_dataset.func[0])
roi_time_series.shape
```

```python
import pandas as pd

pd.read_table(development_dataset.confounds[0]).head()
```

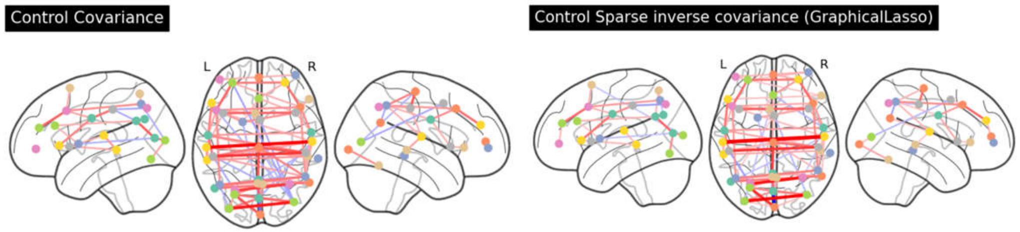
```python
corrected_roi_time_series = masker.transform(
    development_dataset.func[0], confounds=development_dataset.confounds[0])
corrected_correlation_matrix = correlation_measure.fit_transform(
    [corrected_roi_time_series])[0]
np.fill_diagonal(corrected_correlation_matrix, 0)
plotting.plot_matrix(corrected_correlation_matrix, labels=msdl_atlas.labels,
                     vmax=0.8, vmin=-0.8, colorbar=True)
```

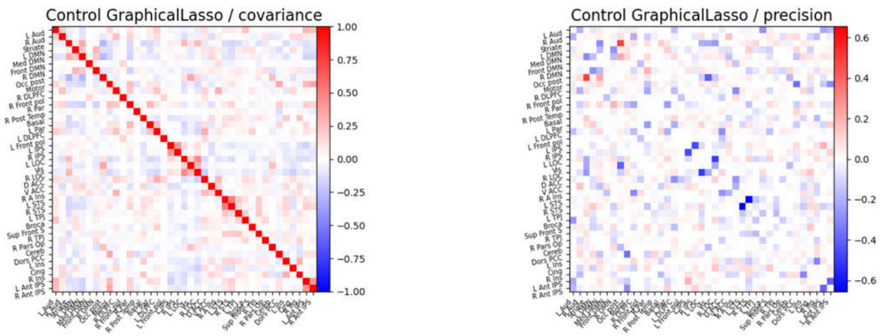
# Nilearn example: functional connectivity

```python
import numpy as np
import matplotlib.pyplot as plt
from nilearn import (datasets, input_data, plotting)
from nilearn.connectome import ConnectivityMeasure

development_dataset = datasets.fetch_development_fmri(n_subjects=30)
msdl_atlas = datasets.fetch_atlas_msdl()

masker = input_data.NiftiMapsMasker(
    msdl_atlas.maps, resampling_target="data",
    t_r=2, detrend=True,
    low_pass=0.1, high_pass=0.01).fit()
correlation_measure = ConnectivityMeasure(kind='correlation')
```

```python
children = []
pooled_subjects = []
groups = []  # child or adult

for func_file, confound_file, phenotypic in zip(
        development_dataset.func,
        development_dataset.confounds,
        development_dataset.phenotypic):

    time_series = masker.transform(func_file, confounds=confound_file)
    pooled_subjects.append(time_series)

    if phenotypic['Child_Adult'] == 'child':
        children.append(time_series)

    groups.append(phenotypic['Child_Adult'])

print('Data has {0} children.'.format(len(children)))
```

```python
from sklearn.metrics import accuracy_score
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.svm import LinearSVC

kinds = ['correlation', 'partial correlation', 'tangent']
_, classes = np.unique(groups, return_inverse=True)
cv = StratifiedShuffleSplit(n_splits=15, random_state=0, test_size=5)
pooled_subjects = np.asarray(pooled_subjects)
```

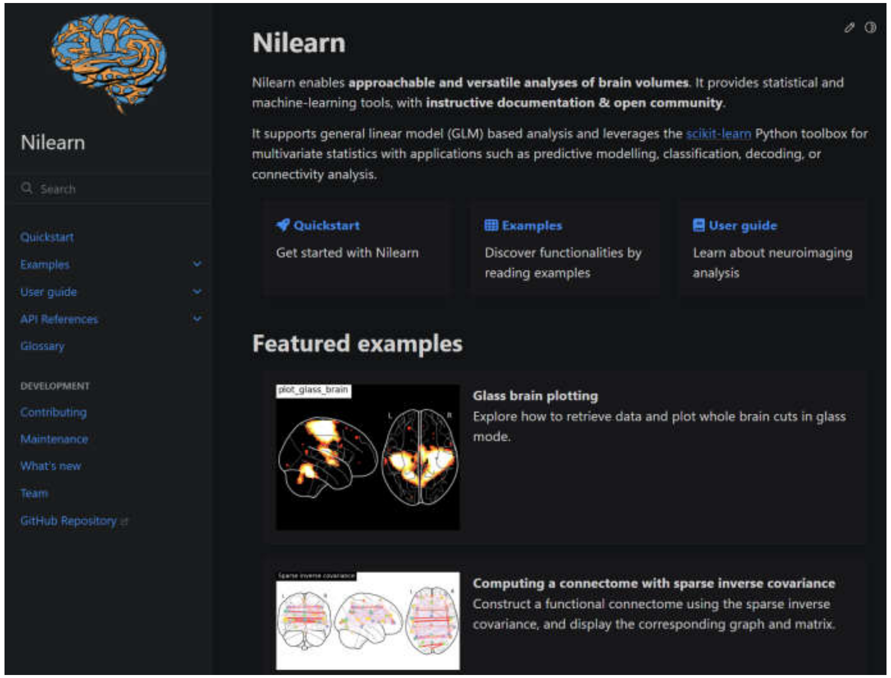



```python
scores = {}
for kind in kinds:
    scores[kind] = []
    for train, test in cv.split(pooled_subjects, classes):
        # *ConnectivityMeasure* can output the estimated subjects coefficients
        # as a 1D arrays through the parameter *vectorize*.
        connectivity = ConnectivityMeasure(kind=kind, vectorize=True)
        # build vectorized connectomes for subjects in the train set
        connectomes = connectivity.fit_transform(pooled_subjects[train])
        # fit the classifier
        classifier = LinearSVC().fit(connectomes, classes[train])
        # make predictions for the left-out test subjects
        predictions = classifier.predict(
            connectivity.transform(pooled_subjects[test]))
        # store the accuracy for this cross-validation fold
        scores[kind].append(accuracy_score(classes[test], predictions))
```

# Takeaways

- Python based env are ready for neuroimaging studies
- Learning by doing and start with the examples
- ***Engineering tools*** vs *Scientific discovery*
  - Interpretability and explainability



# Useful resources

McGill QLS612 course: https://neurodatascience.github.io/QLS612-Overview/

https://inria.github.io/scikit-learn-mooc/ml_concepts/slides.html

https://www.3blue1brown.com/topics/linear-algebra

3b1b Gradient Descent: https://www.youtube.com/watch?v=IHZwWFHWa-w

Python Neuroimaging libs family: https://nipy.org/