

Machine Learning

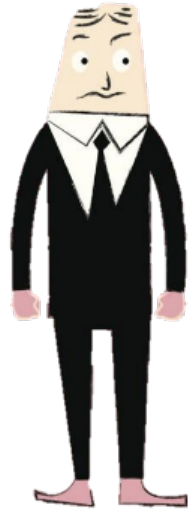
An introduction

by: Estefany Suárez & Jake Vogel

29/06/2021

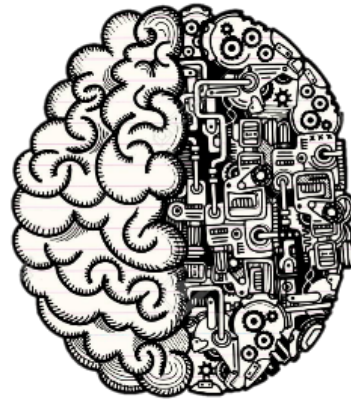
Machine Learning (ML)

Learn from experience

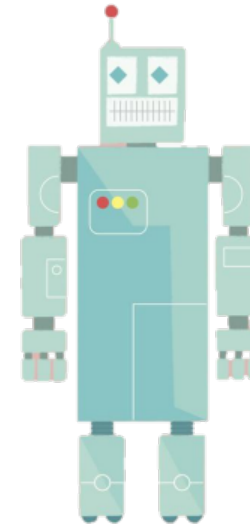


DATA

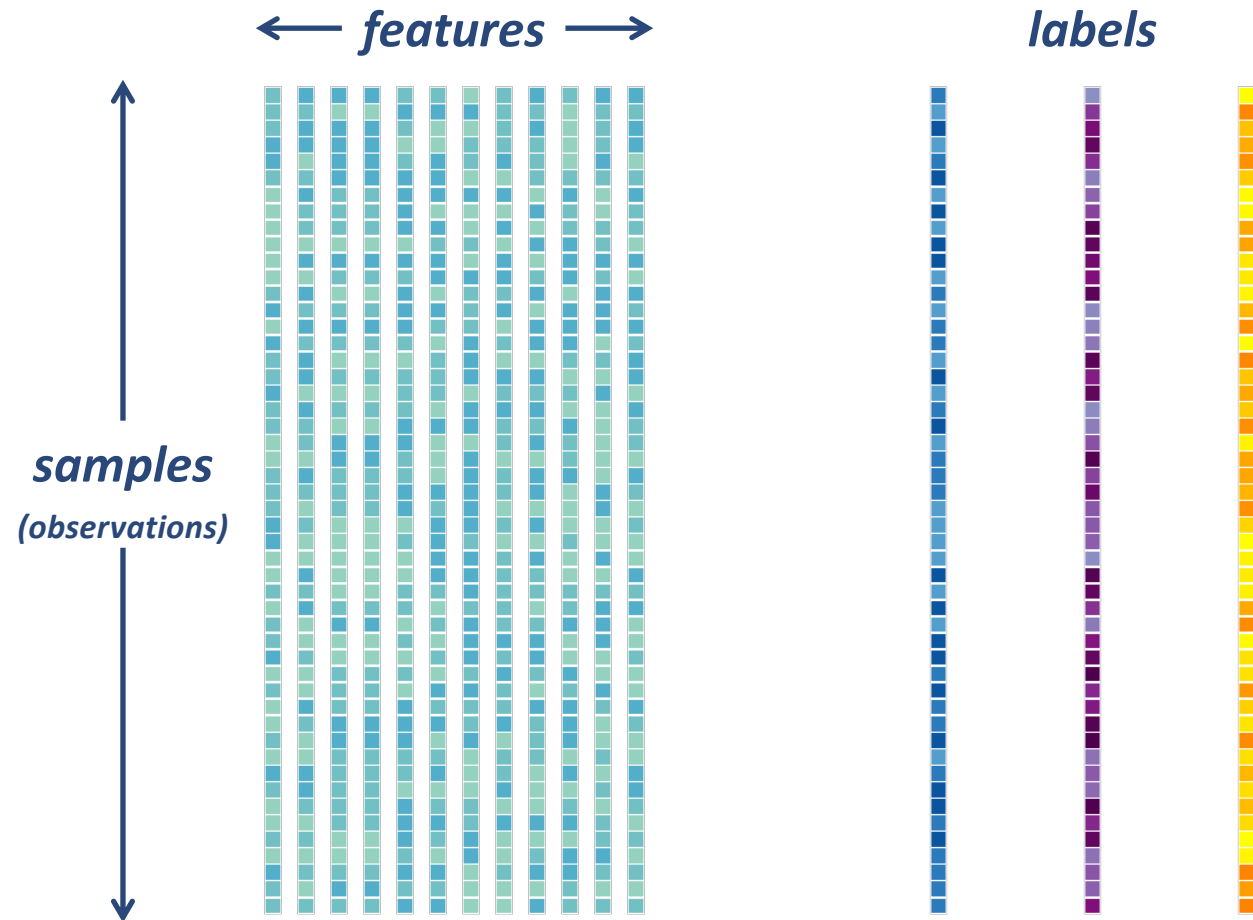
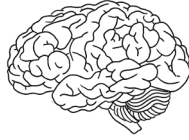
Learn from ~~experience~~



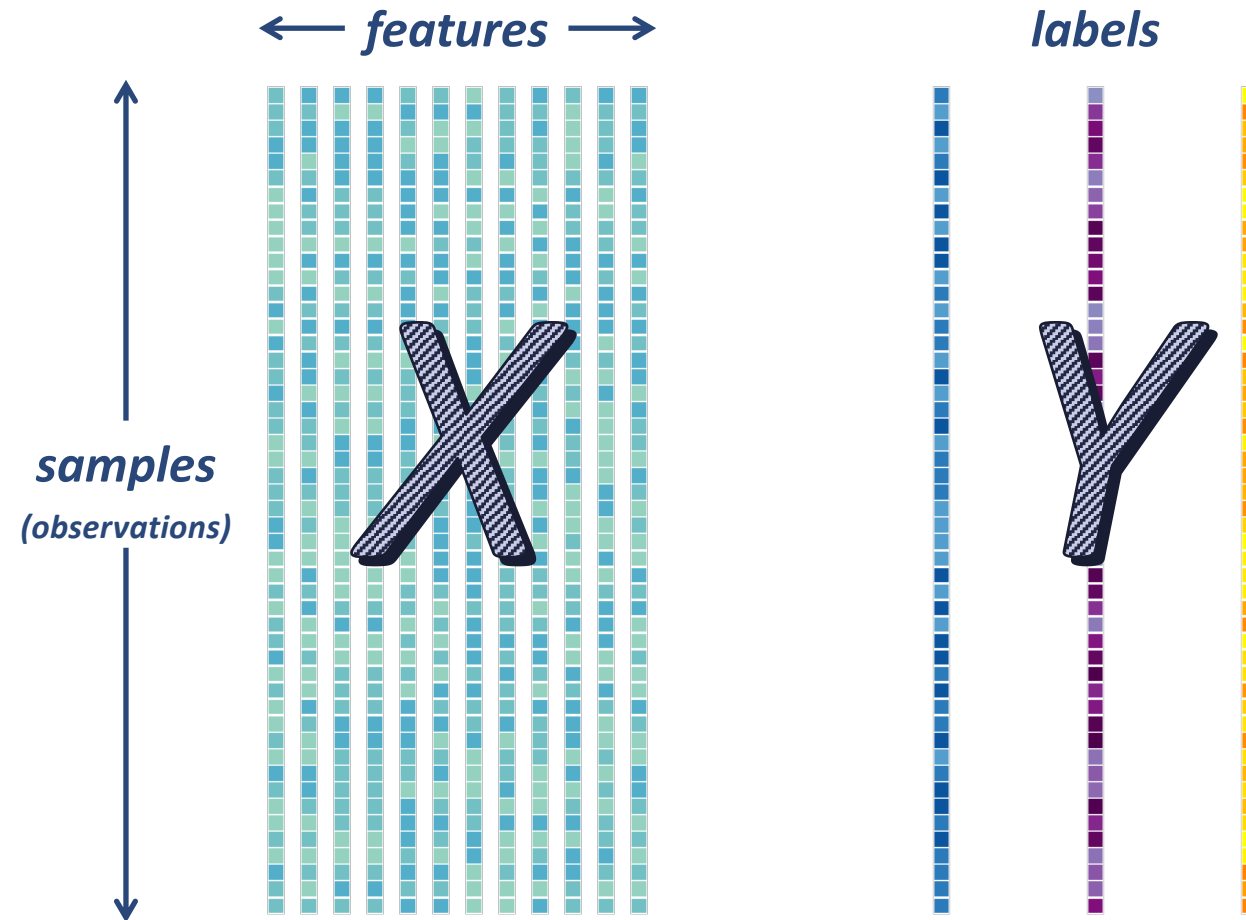
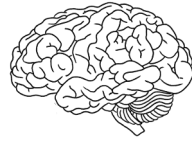
Follow instructions



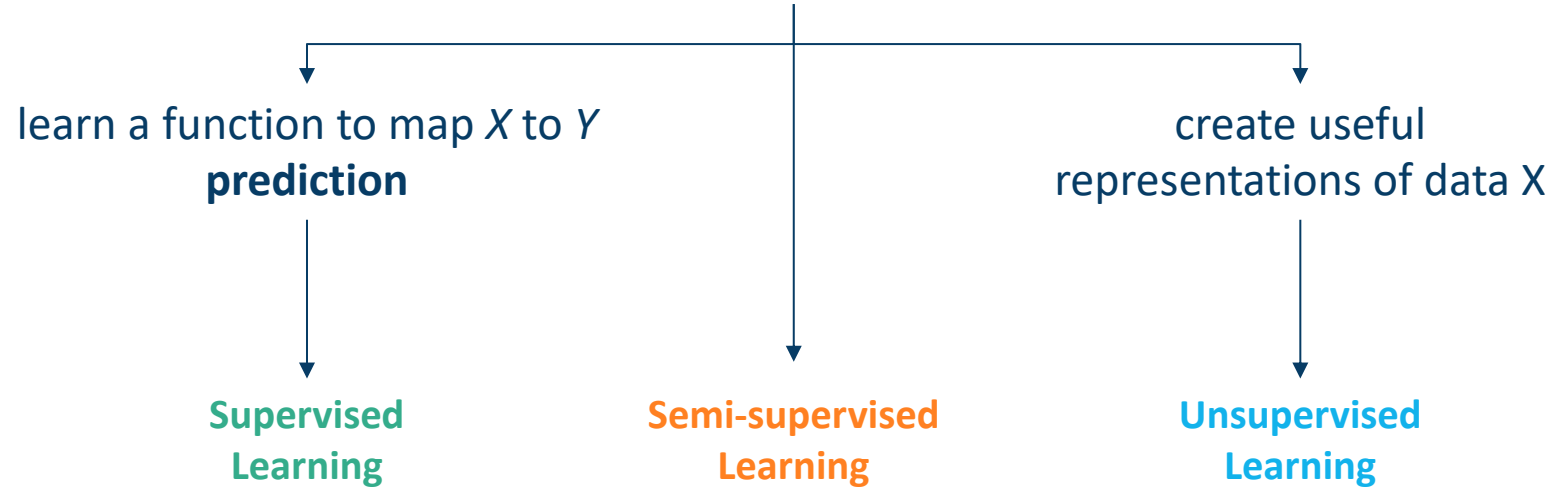
Terminology



Terminology



ML algorithms



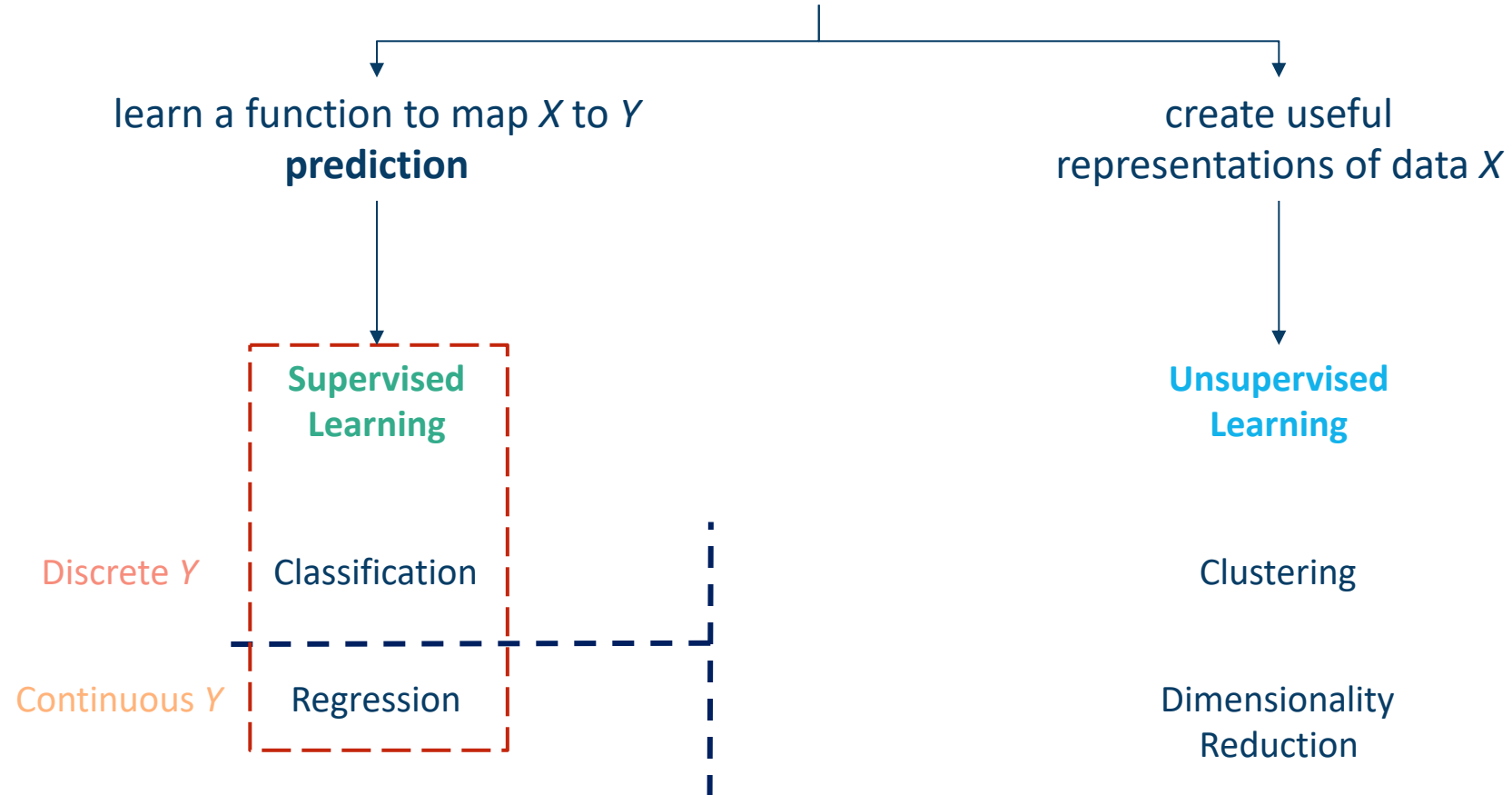
x, y

$f: x \rightarrow y$

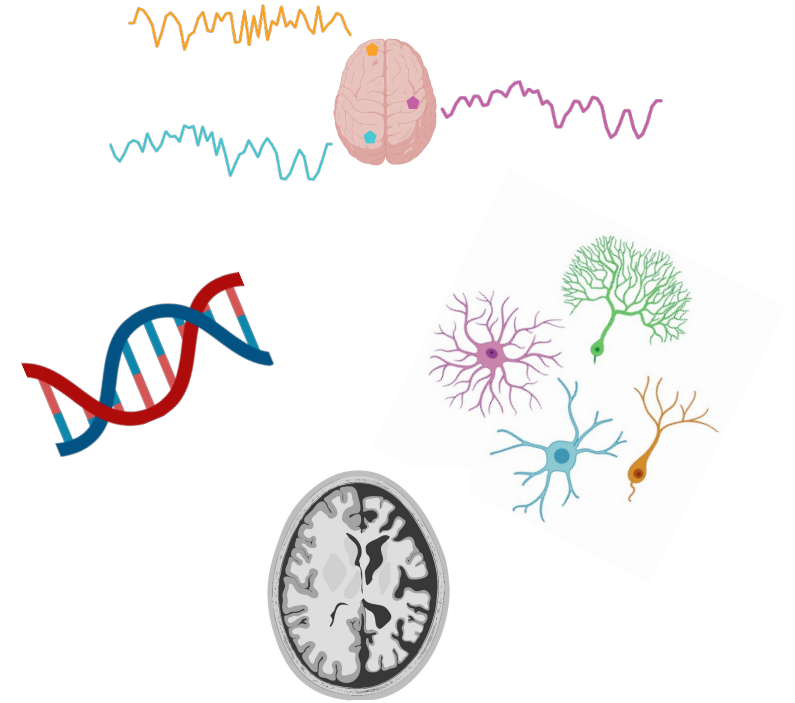
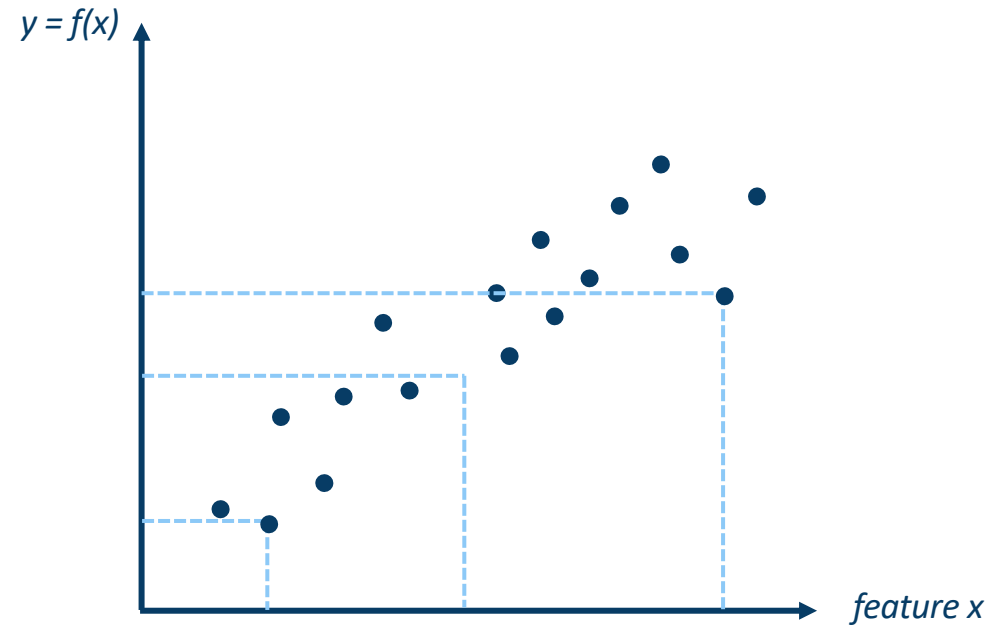
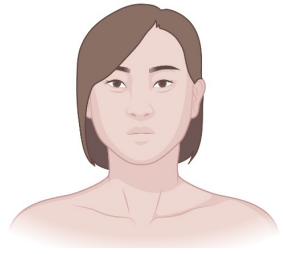
$y = f(x)$

x 

ML algorithms



Regression

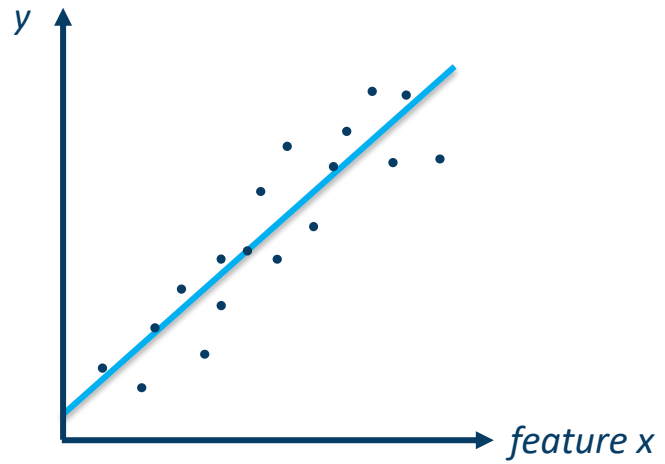


Regression

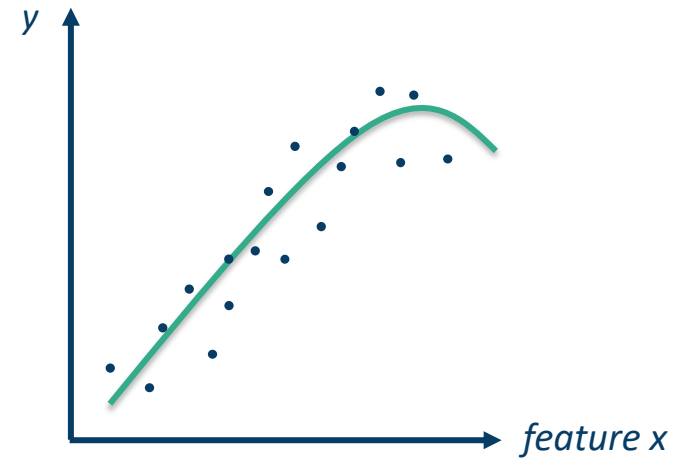
$$f: x \rightarrow y$$

$$y = f(x)$$

$$y = ax + b$$



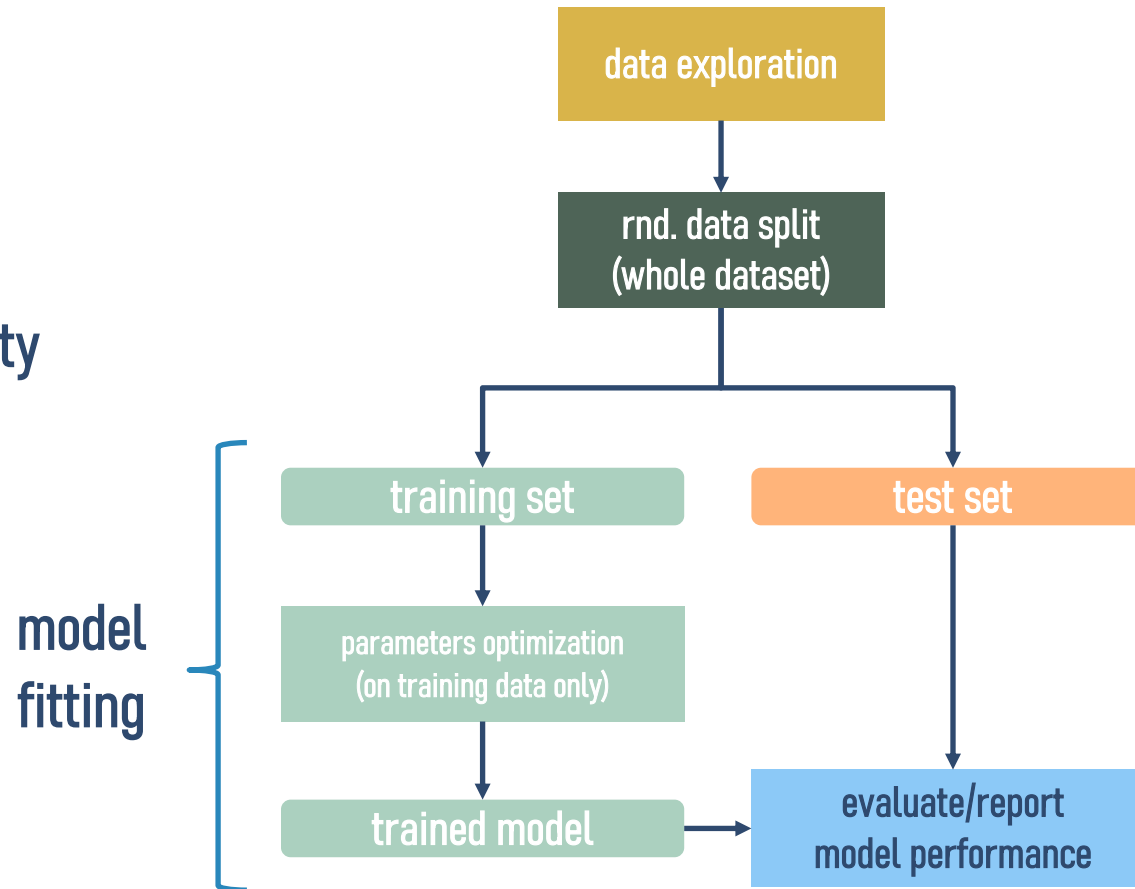
$$y = ax^2 + bx + c$$



Typical ML Pipeline

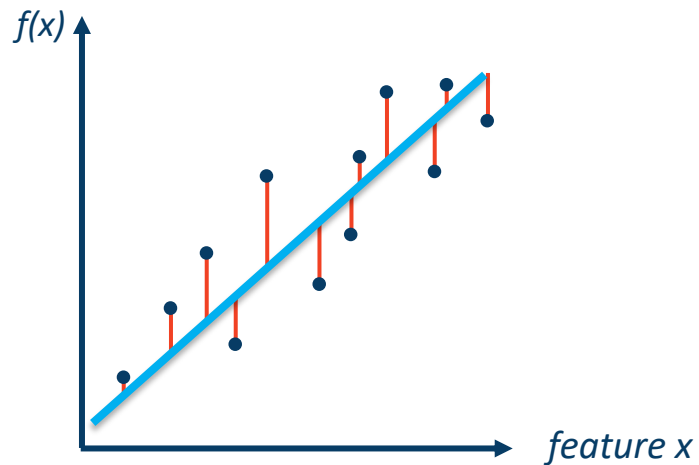
Why cross-validation (CV)?

- Avoid overfitting
- Improve model generalizability



ML basic ingredients

$$f: x \rightarrow y$$
$$y = f(x)$$



①



②

Model or estimator
($f: x \rightarrow y$)

$$\longrightarrow y = ax + b$$

③

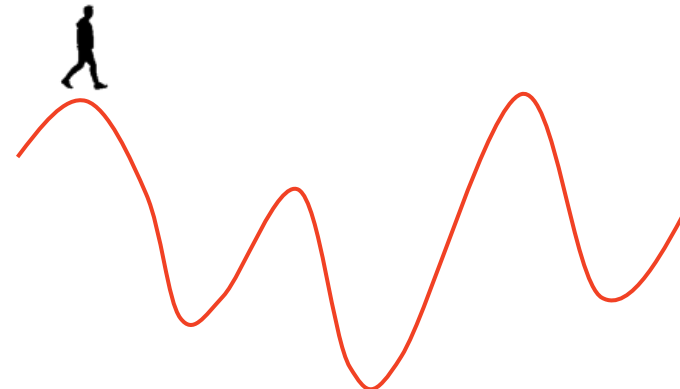
Loss function

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

④

Optimization algorithm

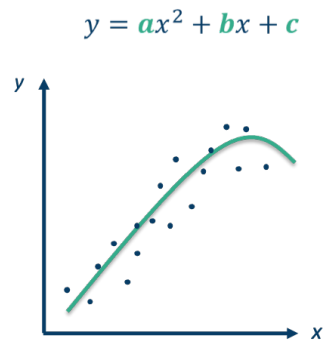
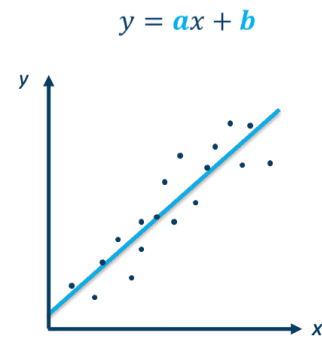
\longrightarrow gradient descent



ML basic ingredients

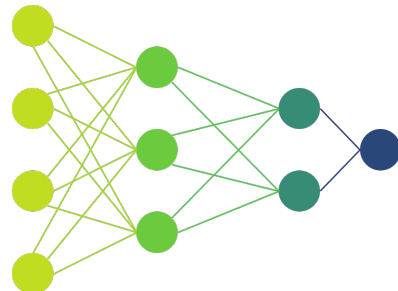
① Model (*estimator*) selection

linear regression



polynomial regression

multi-layer
perceptron
(ANN)



② Loss (*cost*) function

$$\text{mean squared error} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$\text{mean absolute error} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

$$\text{max error} = \max(|y_i - \hat{y}_i|)$$

$$\text{explained var.} = 1 - \frac{\text{Var}(y - \hat{y})}{\text{Var}(y)}$$

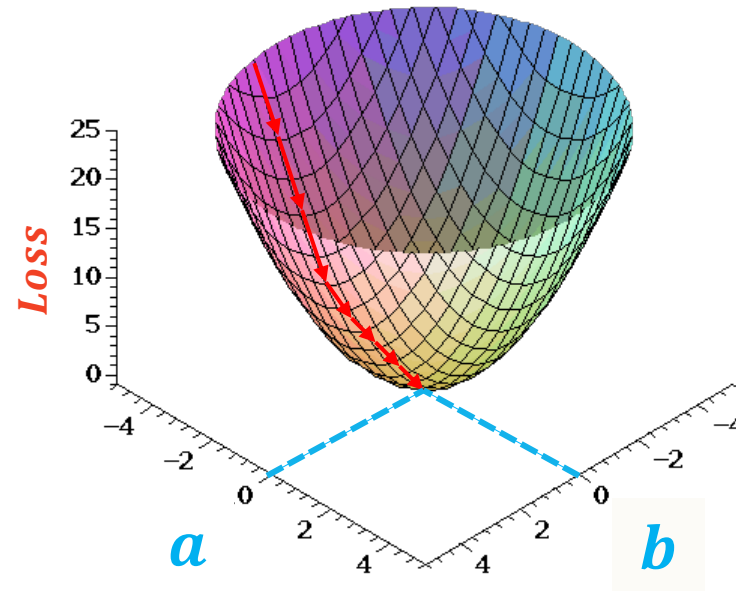
$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

ML basic ingredients

- ③ Optimization algorithm for model parameters!

$$y = ax + b$$

$$\text{Loss} = f(a, b)$$



Gradient Descent

Initialize model parameters (a, b) randomly iterate between:

1) Estimate output $\hat{y}_i = ax_i + b$

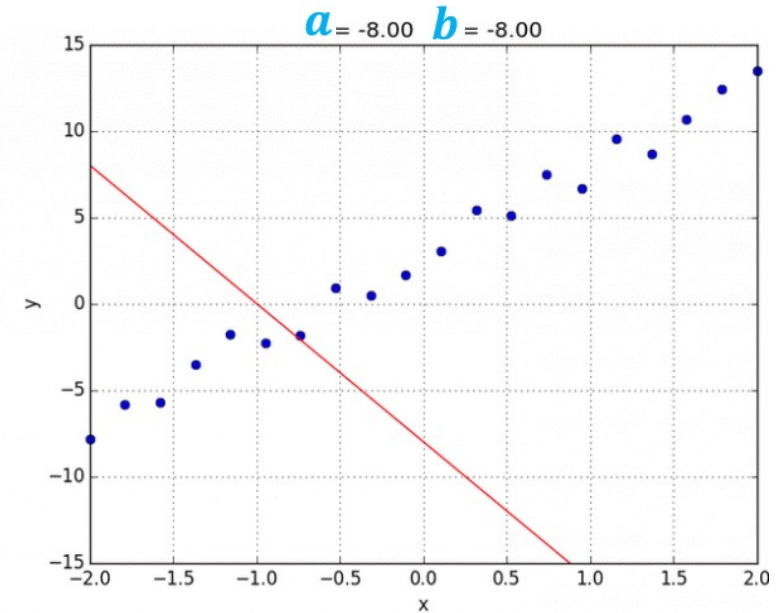
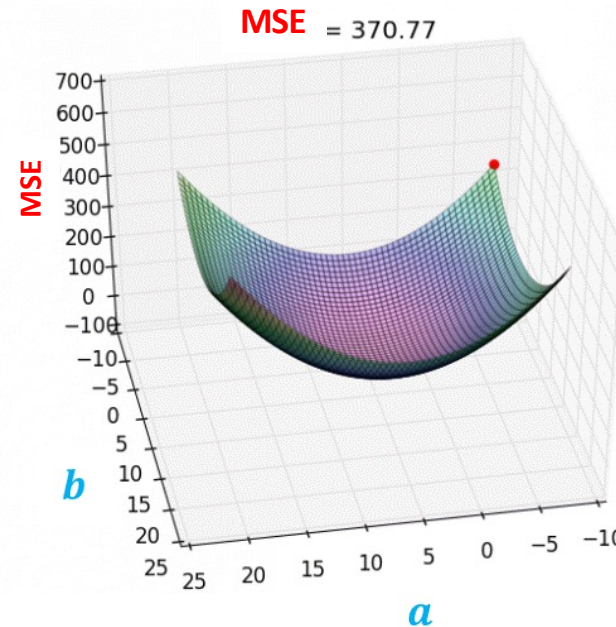
2) Compute *loss* $MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$

3) Compute the gradient

4) Update parameters (a, b)

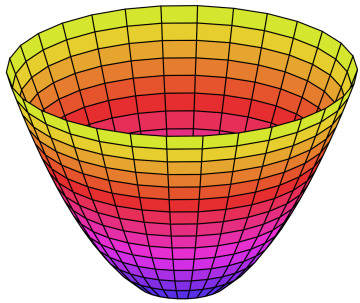
Training data

x_{train} y_{train}



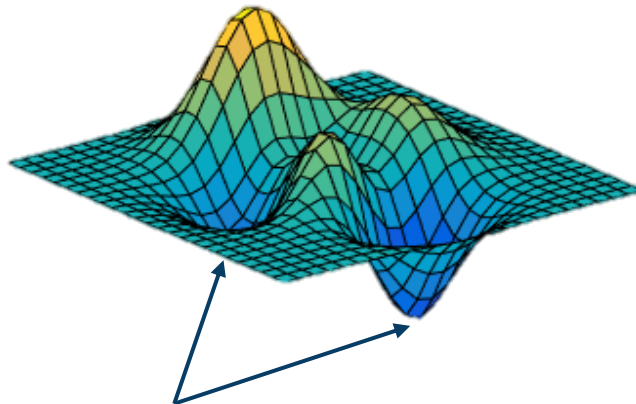
There is a problem though ...

simple loss function



global
minimum

convoluted loss function



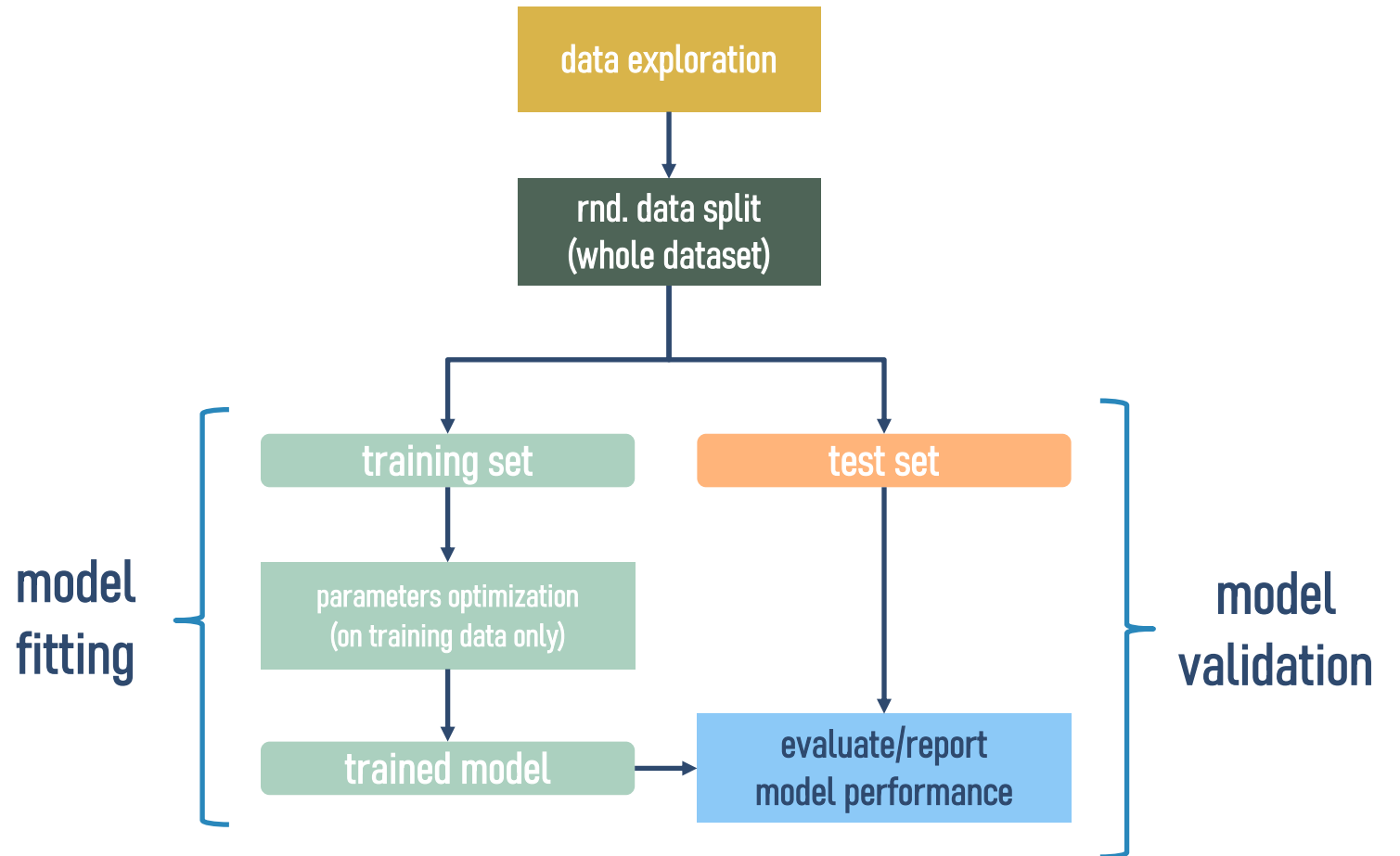
multiple local
minima



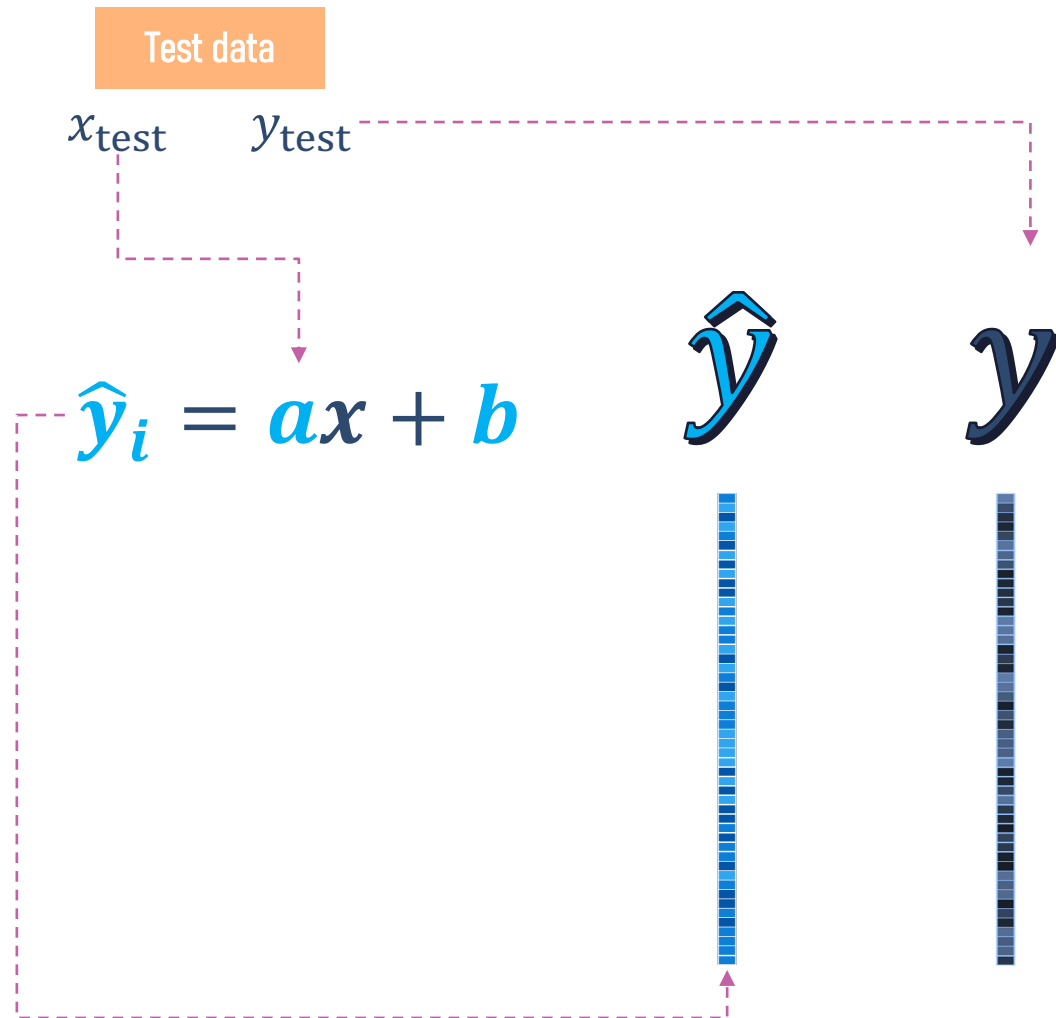
Re-use your data ...

multiple training epochs !

Typical ML Pipeline



Model Validation



score function

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

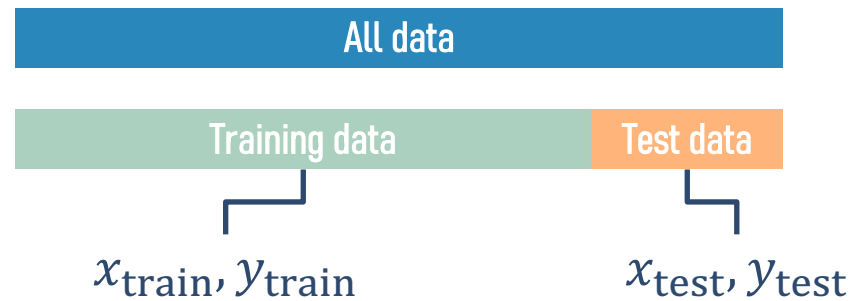
mean absolute error = $\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$

max error = $\max(|y_i - \hat{y}_i|)$

explained var. = $1 - \frac{\text{Var}(y - \hat{y})}{\text{Var}(y)}$

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

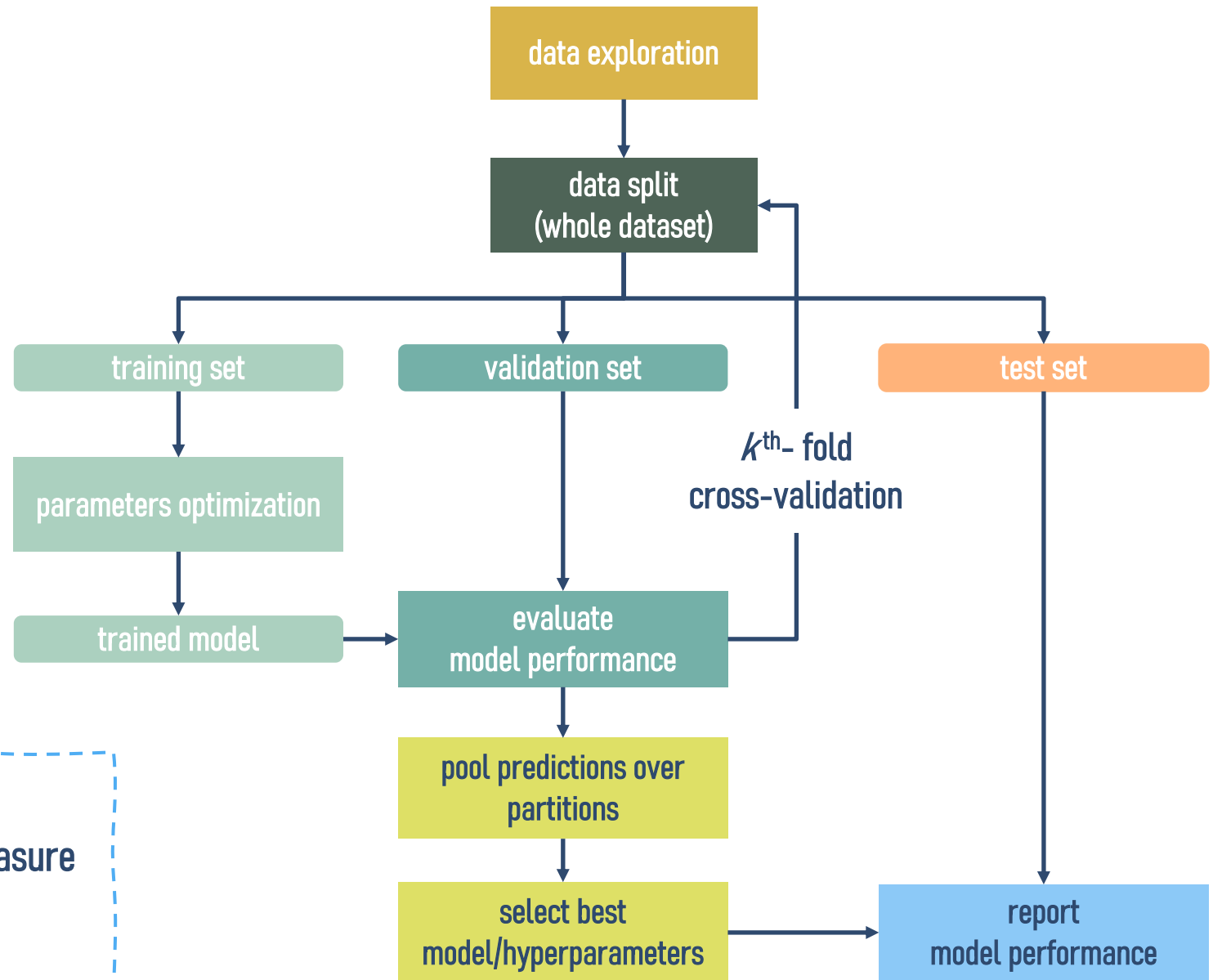
We have yet another problem ...



BUT ... What if your data is very heterogeneous and the “*outlier*” samples/observations happen to be in the test set?

This would greatly affect the generalizability of your model!!!

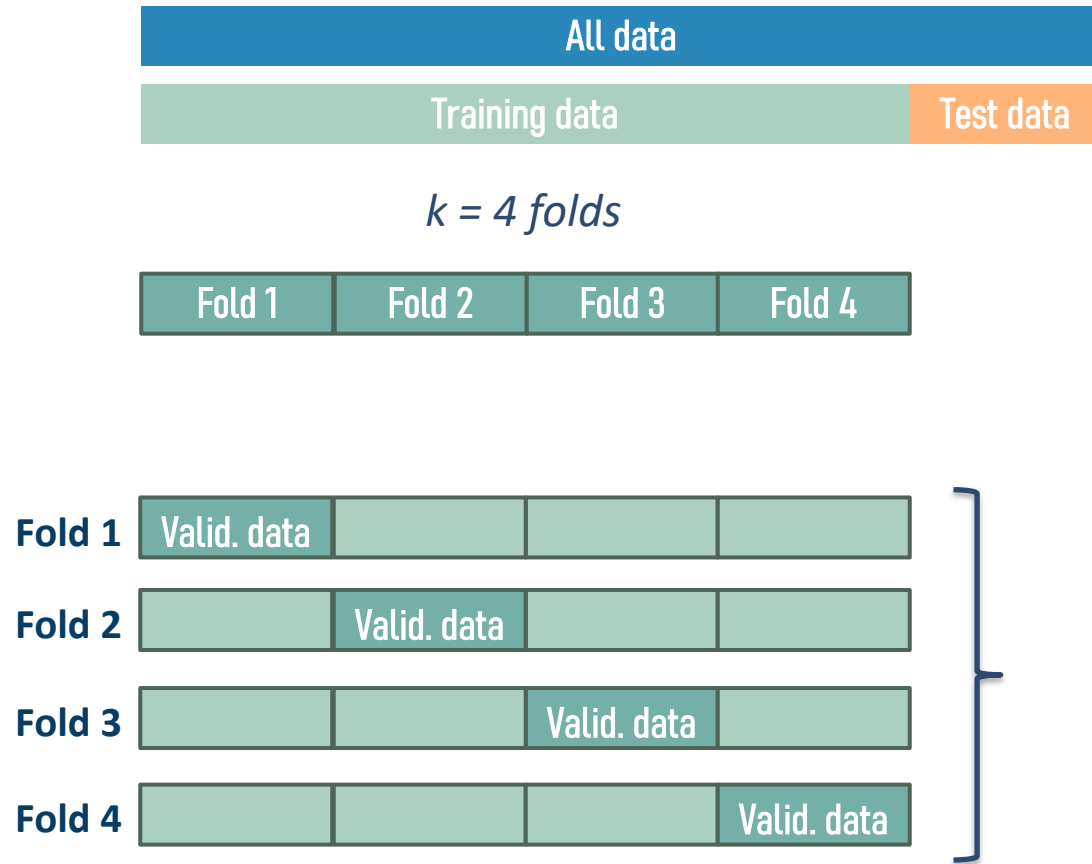
Typical ML Pipeline



Why k^{th} -fold CV?

- Provide a more accurate performance measure
[average, standard deviation, range]
- Model selection
- *Hyper*-parameter tuning

[*k-fold*] Cross-Validation (CV)



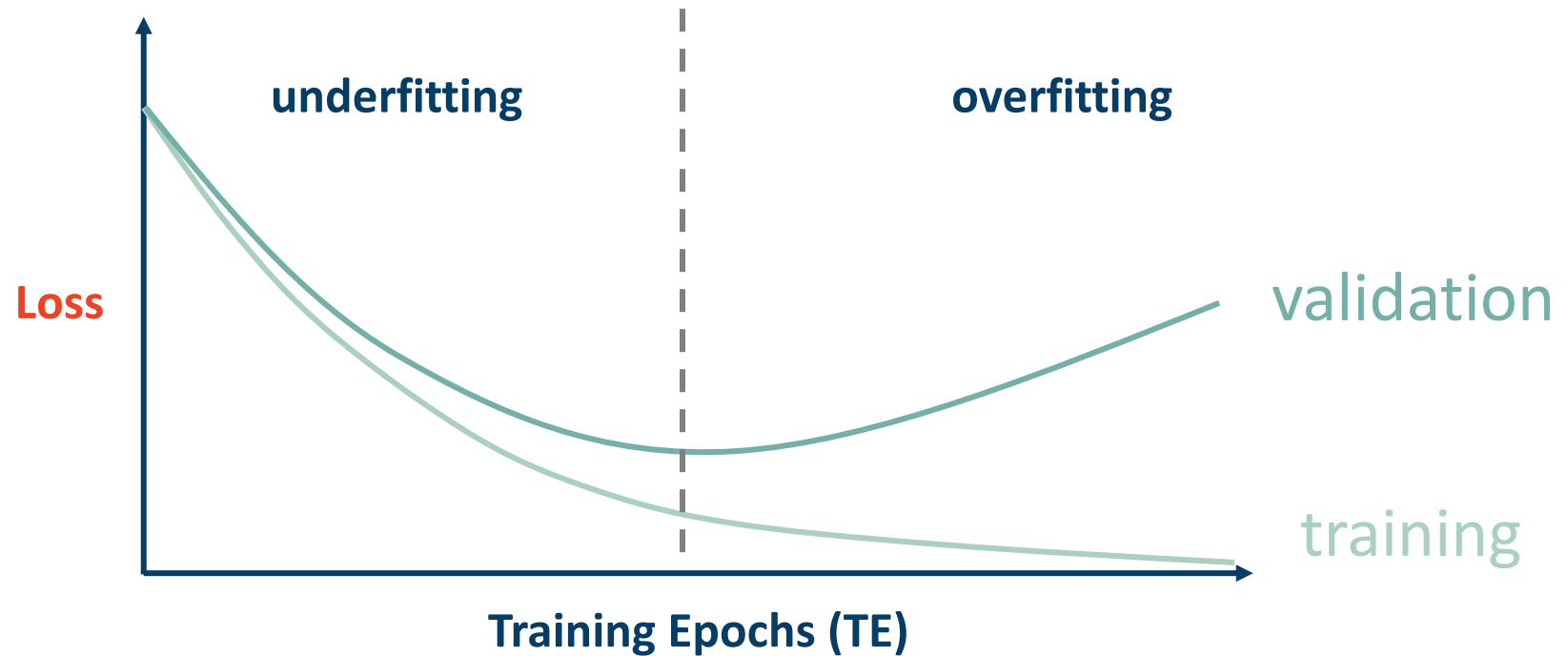
1. For each fold:
 - I. Train model parameters with training set
 - II. Evaluate training with validation set
2. Based on validation set, select *optimal* model and hyper-parameters
3. Only at the very, very **END**, report error on test set

Diagnosing overfitting with *Learning Curves*

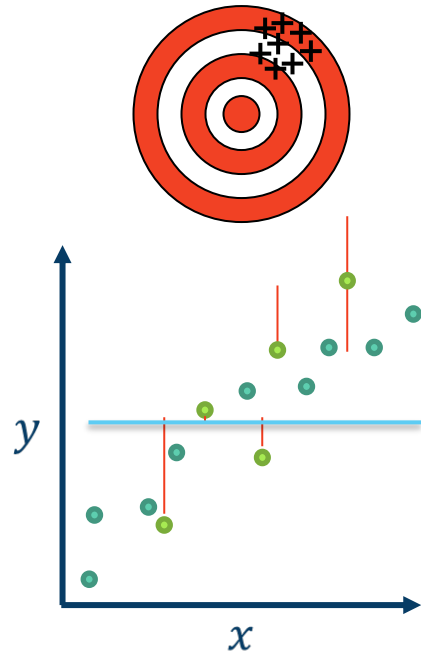
k^{th} -fold



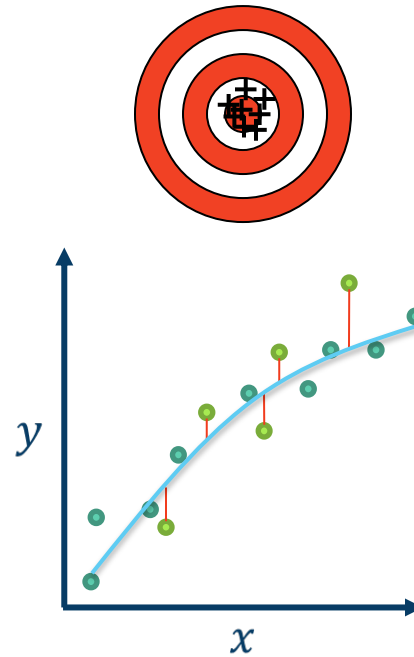
How many TE?



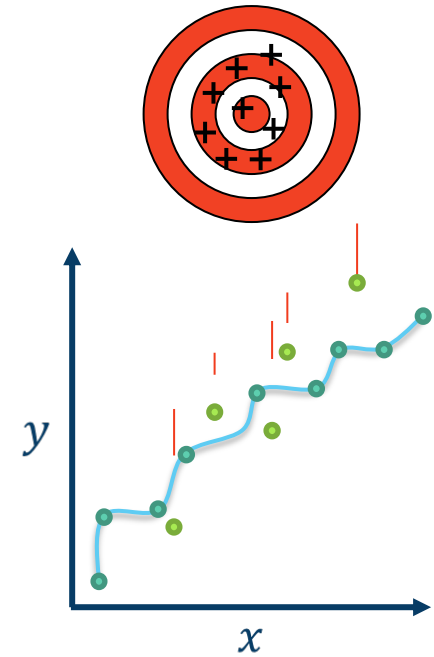
Bias-Variance Trade-Off



Underfitting



Optimum

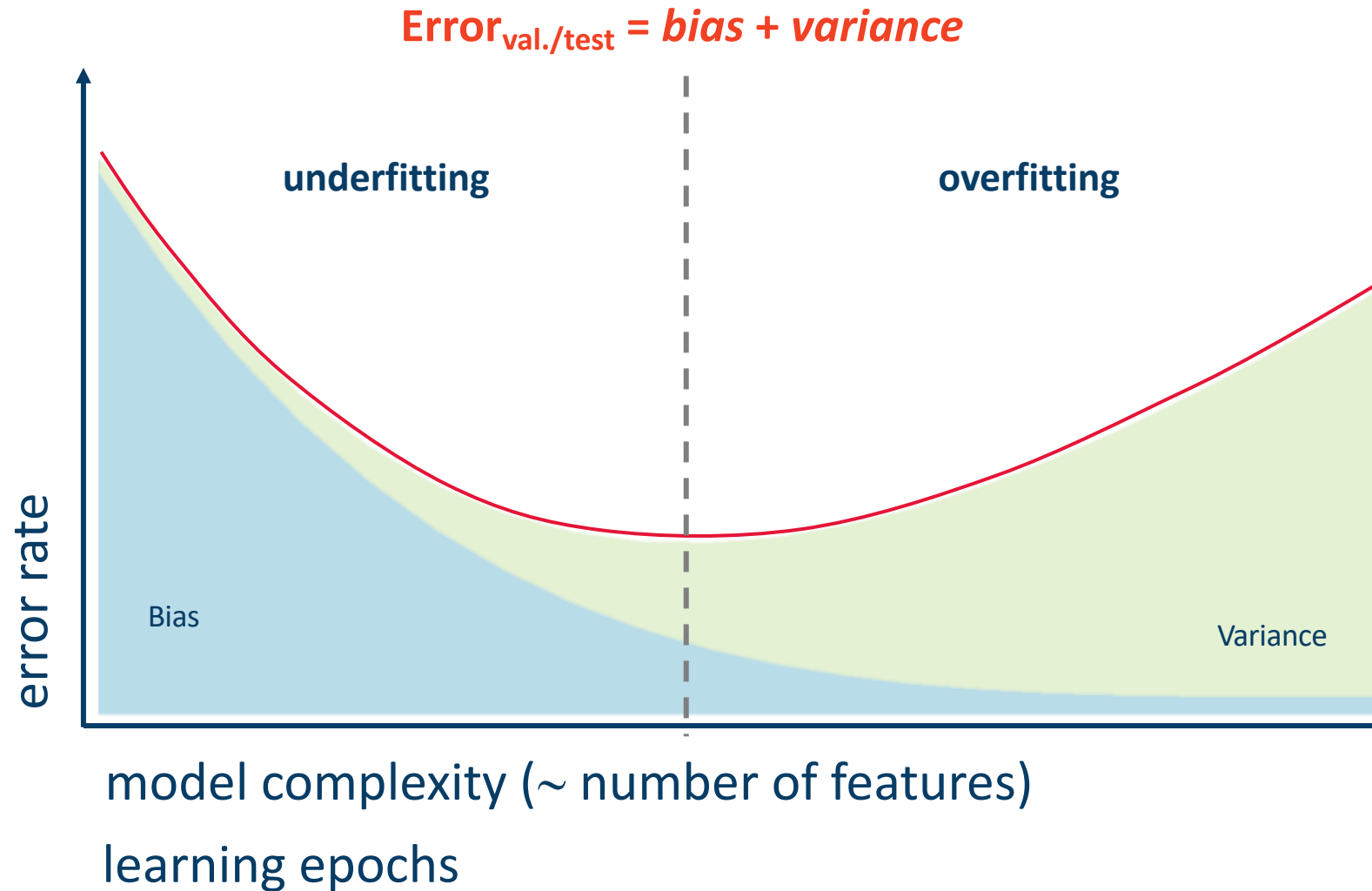


Overfitting

● Training set

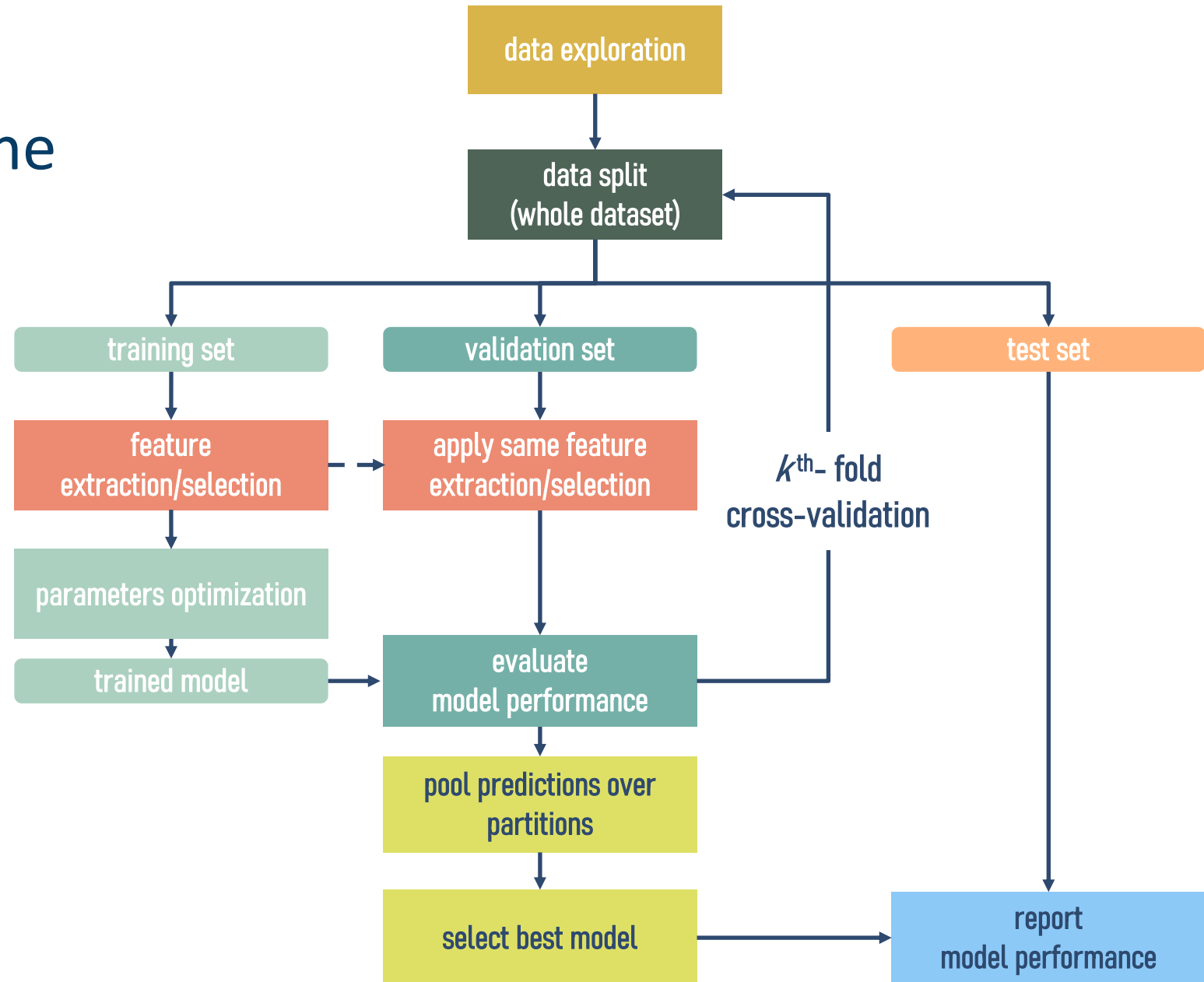
● Test set

Bias-Variance Trade-Off



Typical ML Pipeline

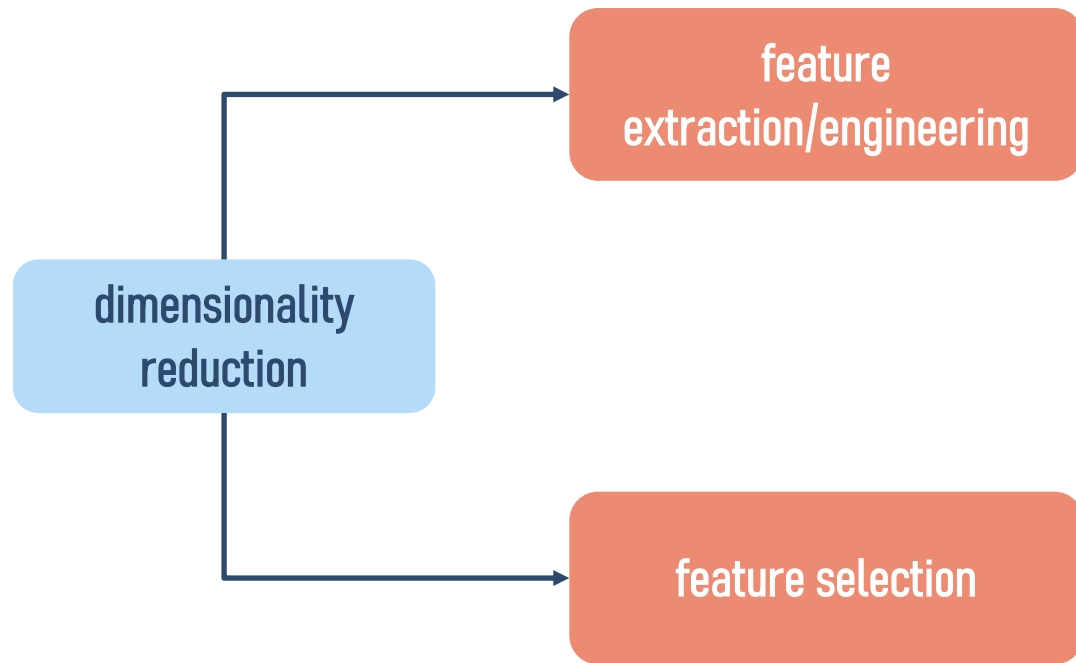
dimensionality
reduction



WHY DR ?

- Curse of dimensionality (more features than samples)
- Intrinsic dimension may actually be small (redundant data)
- Extract “salient” features
- Remove noisy features
- VISUALIZATION!!!!

Tip 1: Dimensionality Reduction



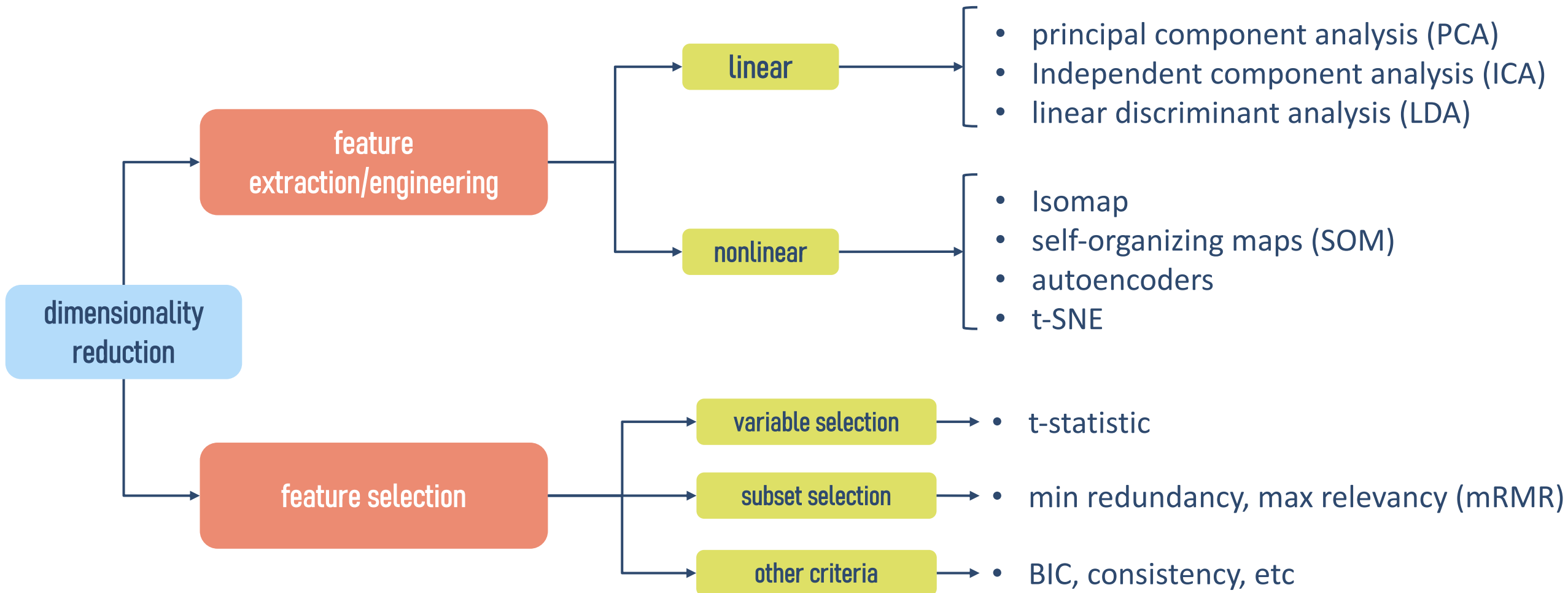
- Compact representation of the data
- Maps input features into a lower dimensional space

e.g. If features are $X = [x_1, x_2, x_3, x_4]$
then $Z = T(X) = [c_1x_1 + c_2x_2, \quad x_3 * x_4]$

- Selection of a subset of input features
- Features are still in original space

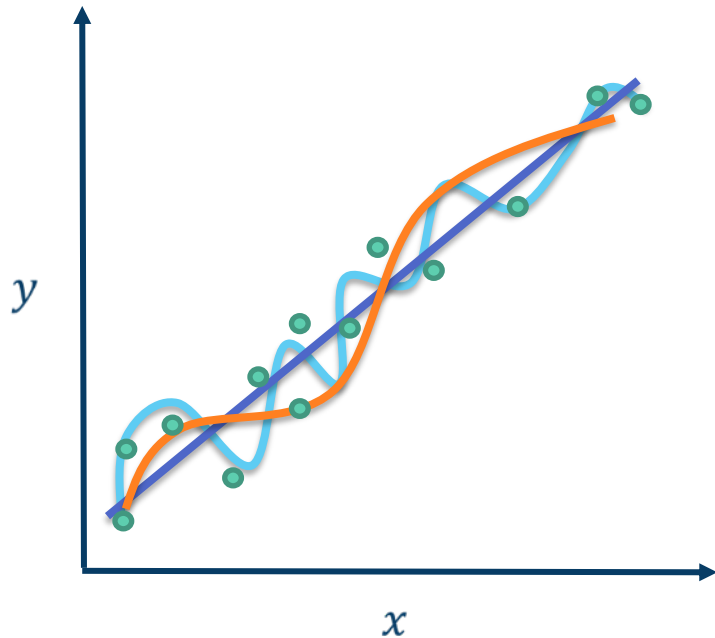
e.g. $Z = S(X) = [x_2, x_3]$

Tip 1: Dimensionality Reduction



Tip 2: Regularization

$$y = \cancel{\beta_0} + \beta_1 x + \cancel{\beta_2} x^2 + \cdots + \cancel{\beta_p} x^p$$



Penalties on the **LOSS** function to prevent overfitting!

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

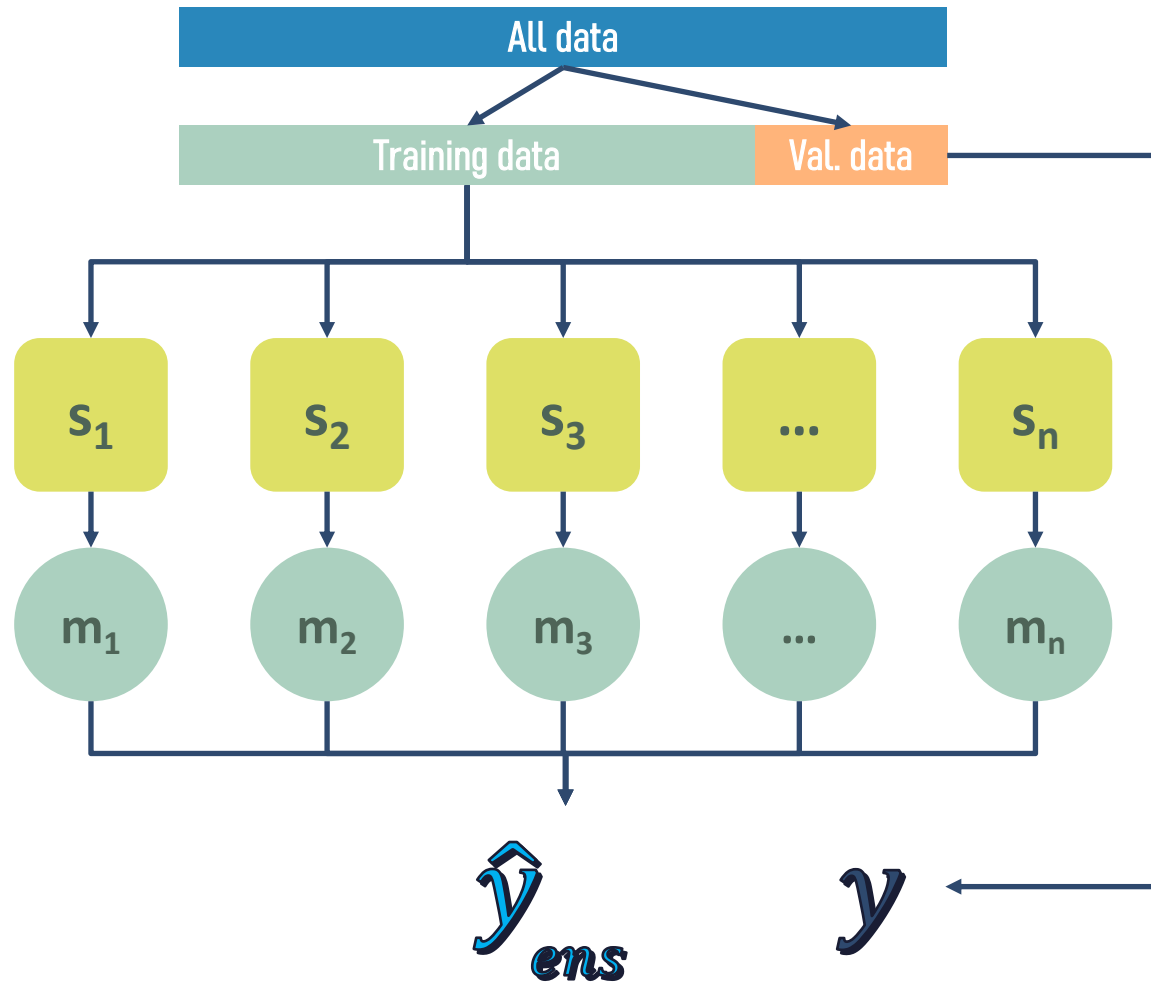
$$MSE = \sum_{i=1}^n \left(y_i - \sum_{j=1}^p x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p |\beta_j|$$

- 2) L2/Ridge: constrains parameters to be **small**

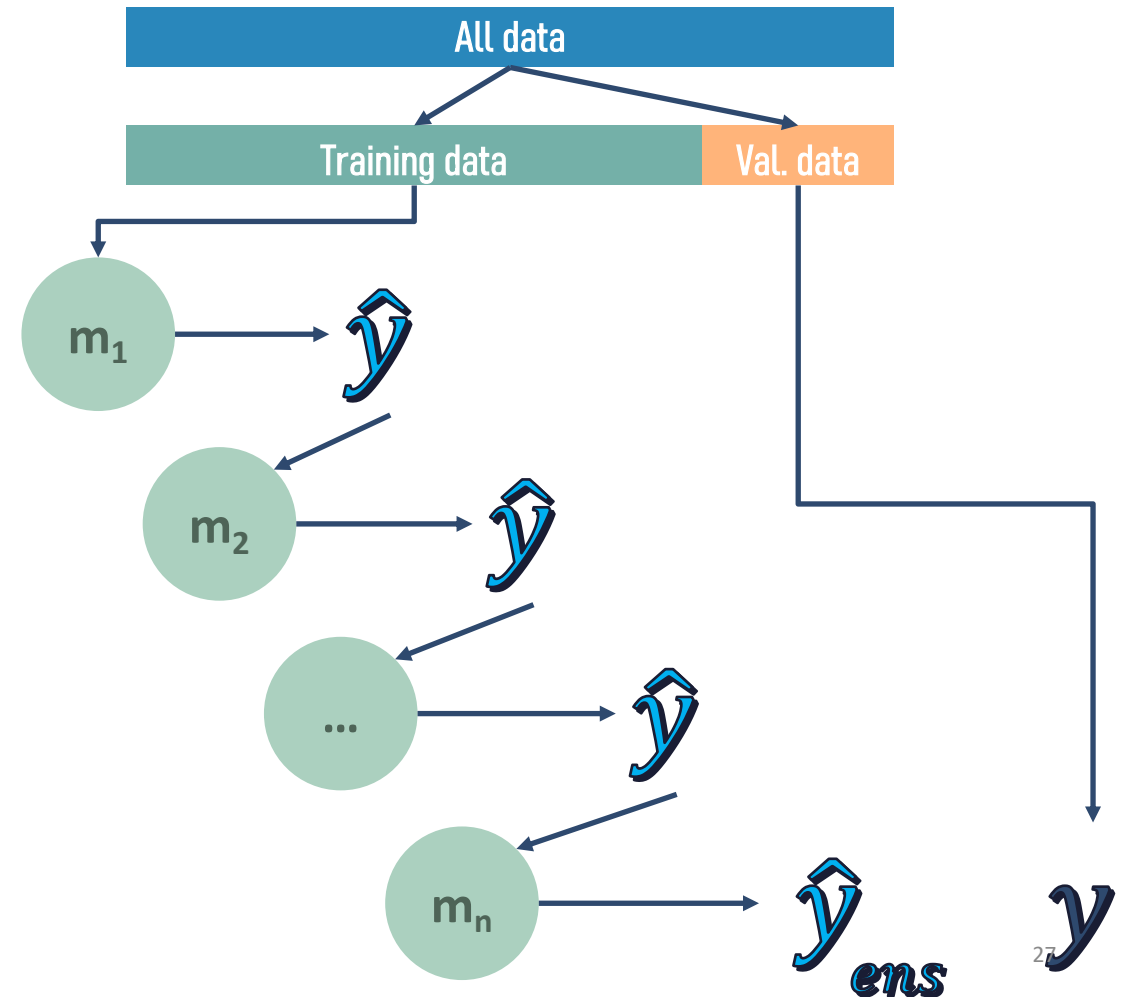
$$MSE = \sum_{i=1}^n \left(y_i - \sum_{j=1}^p x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

Tip 3: Ensemble Methods

bagging = bootstrapp resampling + aggregation



boosting



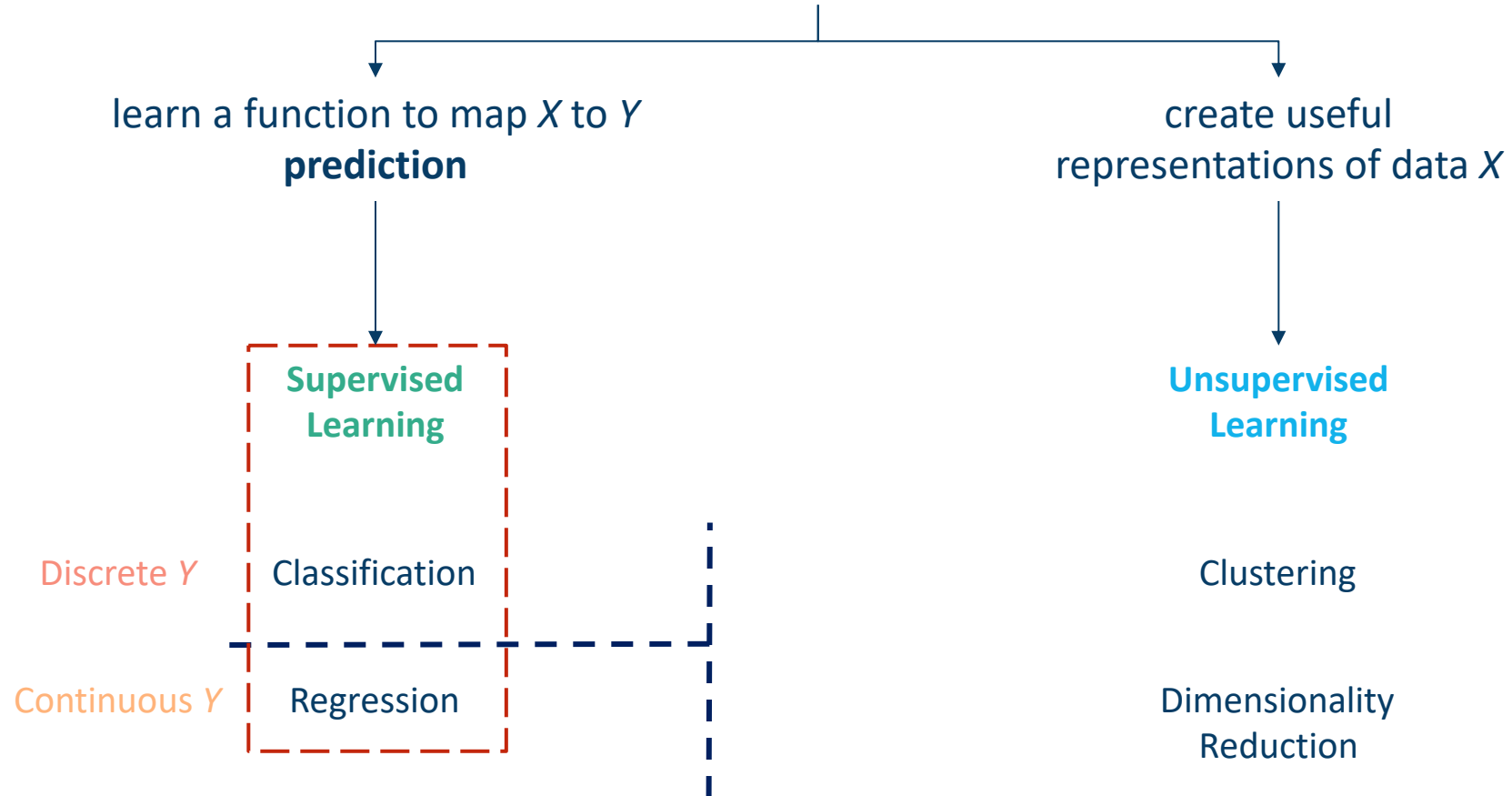
Sources of Bias

Type	DO NOT ☹️	Sexy name	DO 😊
<i>k</i> -hacking	Try many <i>k</i>'s in <i>k</i>-fold CV (or different training %) and report only the best	<i>k</i> -hacking	Pick $k=10$, repeat It many times (>200 or as many as possible!), and report the full distribution (NOT boxplots!)
<i>metric</i> -hacking	Try different performance metrics (e.g., accuracy, F1, AUC, error rate, etc.) and report the best	<i>m</i> -hacking	Choose the most appropriate and recognized metric for the problem (e.g., AUC for binary classification)
<i>feature/dataset</i> -hacking	Try subsets of feature(s) or subsamples of dataset(s) , but report only the best	<i>d</i> -hacking	Use and report on everything: all analyses on all datasets

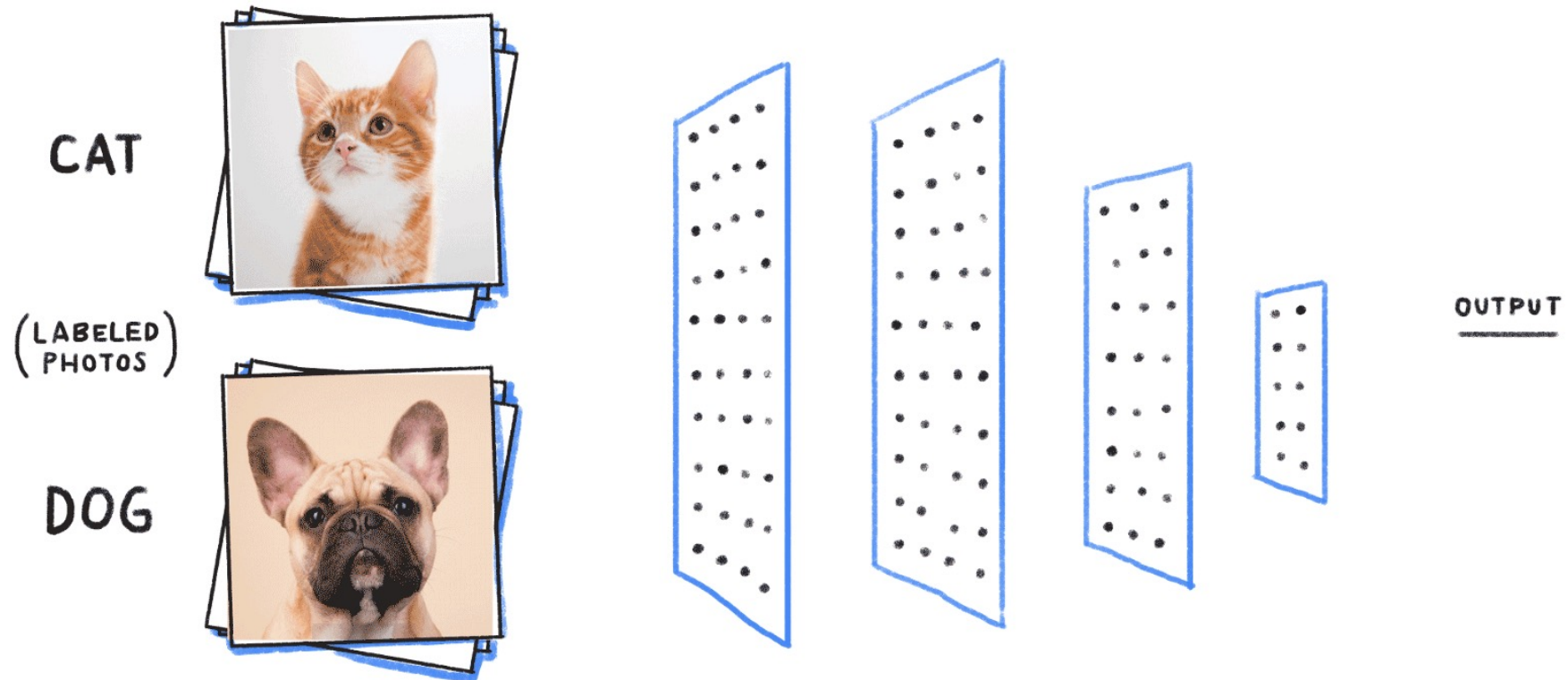
Machine Learning tools for neuroimaging ... in Python!



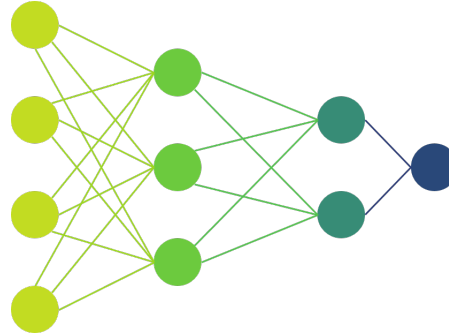
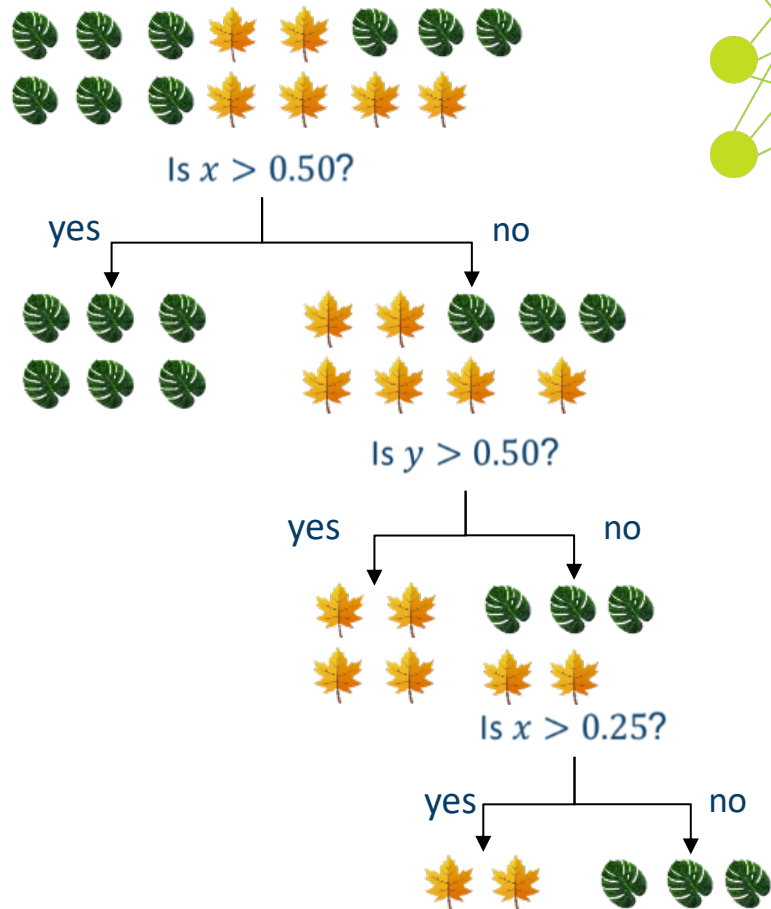
ML algorithms



Classification



Classification

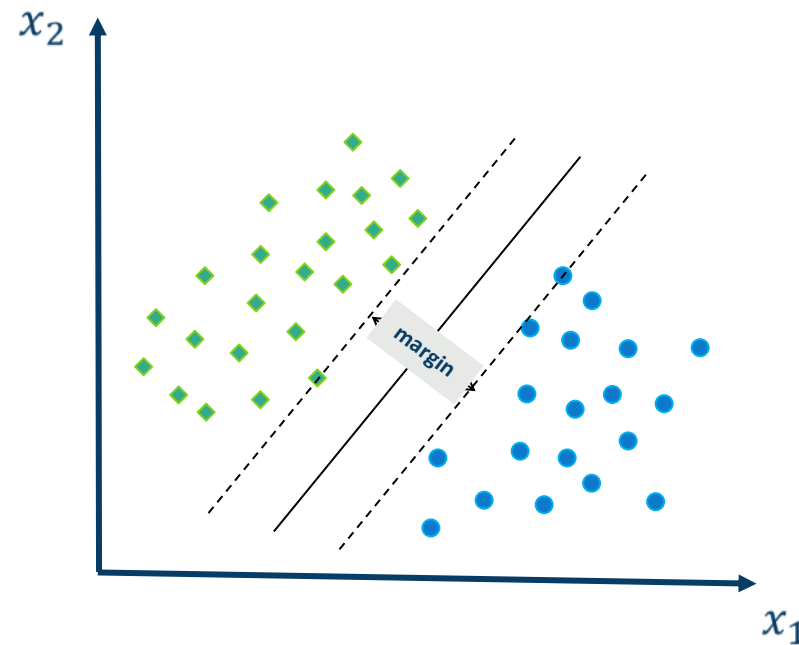


- Support Vector Machine

(SVM)

- Artificial Neural Networks
- Logistic regression
- Decision Trees
- Random Forests

probabilistic
classifiers



Performance Metrics – Binary Classification

confusion matrix

		True condition	
		POSITIVE	NEGATIVE
Predicted condition	POSITIVE	TRUE positive - T_p	FALSE positive - F_p (Type I error)
	NEGATIVE	FALSE negative - F_n (Type II error)	TRUE negative - T_n

score function

$$accuracy = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}$$

$$precision = \frac{T_p}{T_p + F_p}$$

$$recall = \frac{T_p}{T_p + F_n}$$

*Appropriate
when classes
are imbalanced!*

$$F1\ score = \frac{2T_p}{2T_p + F_p + F_n}$$

Multiclass Prediction (\neq Multilabel Prediction)

		True class		
		CAT	DOG	BIRD
Predicted class	CAT	13	0	0
	DOG	0	10	6
	BIRD	0	0	9

- To extend a binary metric to multiclass problems, the data is treated as a collection of binary problems, one for each class.
- The binary metric is then averaged across the set of classes, each of which may be useful in some scenario.

Performance Metrics – Binary Classification

		True condition	
		POSITIVE	NEGATIVE
Predicted condition	POSITIVE	TRUE positive - T_p	FALSE positive - F_p (Type I error)
	NEGATIVE	FALSE negative - F_n (Type II error)	TRUE negative - T_n

Sensitivity (or recall)

$$TPR = \frac{T_p}{P} = \frac{T_p}{T_p + F_n}$$

$$FPR = \frac{F_p}{N} = \frac{F_p}{T_n + F_p}$$

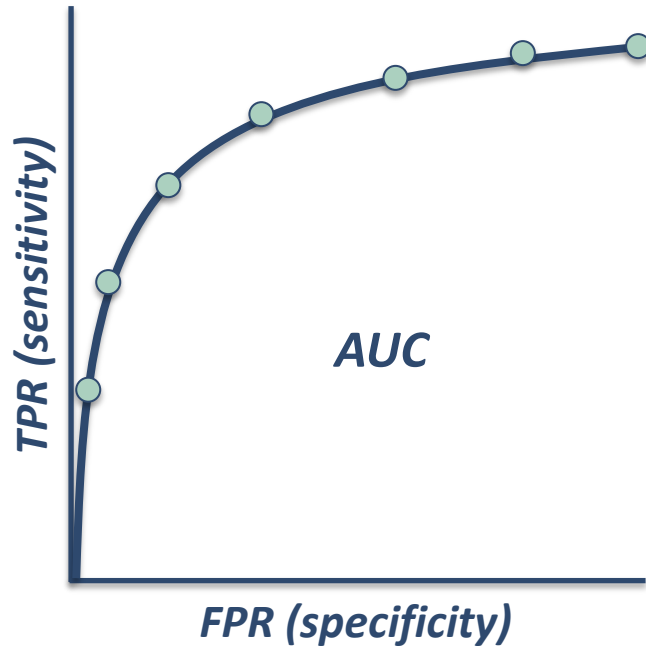
Specificity (or selectivity)

$$FNR = \frac{F_n}{P} = \frac{F_n}{T_p + F_n} \quad TNR = \frac{T_n}{N} = \frac{T_n}{T_n + F_p}$$

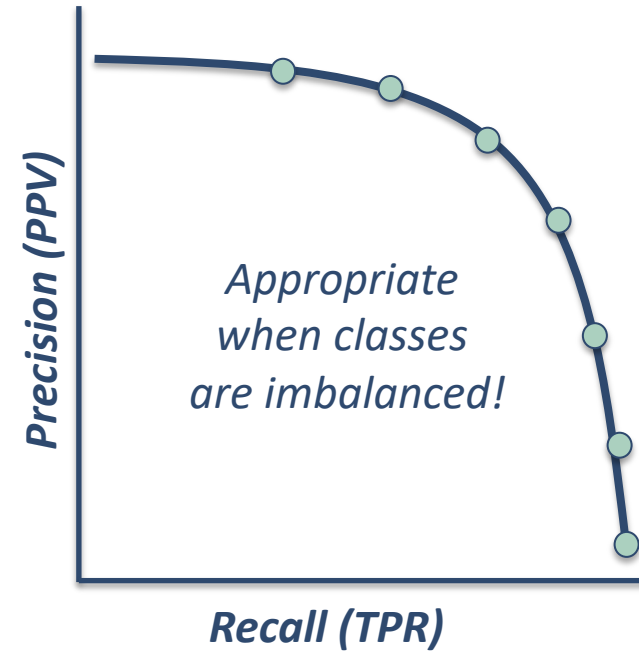
Performance Metrics – Binary Classification

(probabilistic classifiers)

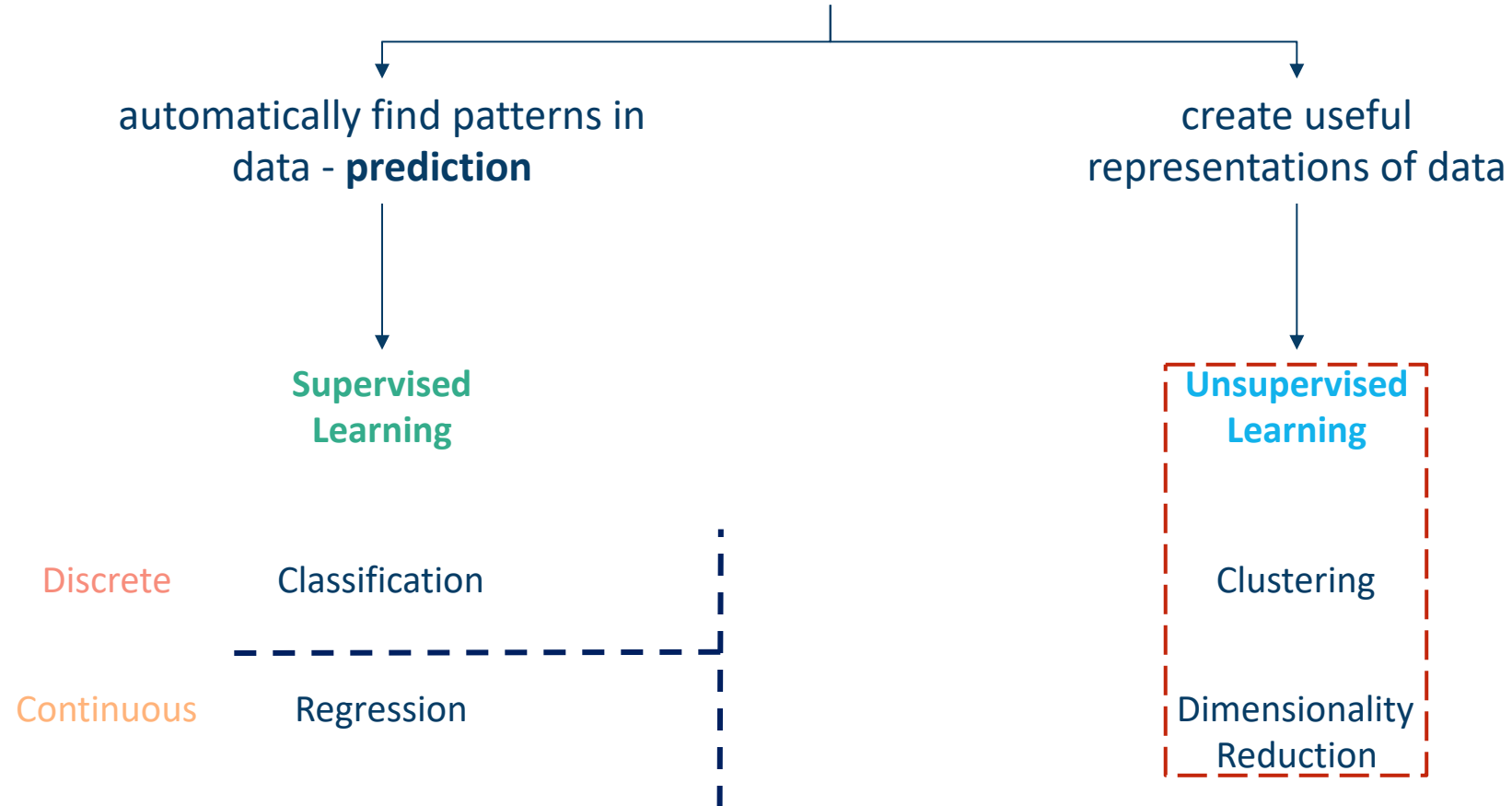
ROC curve



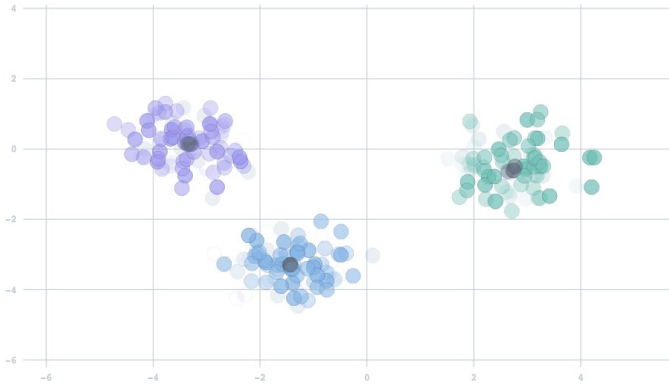
precision-recall curve



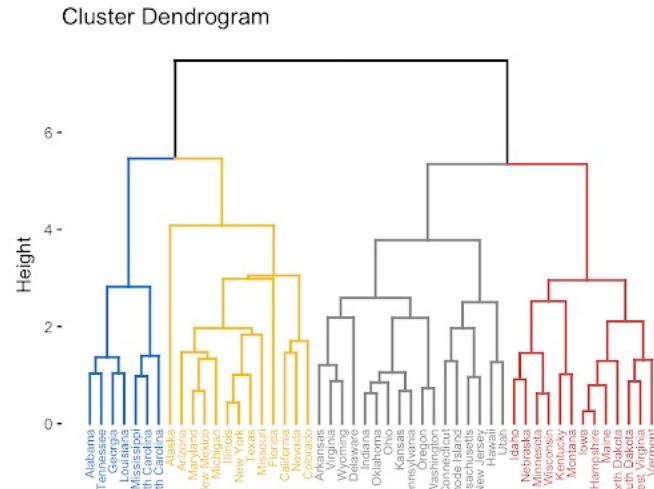
ML algorithms



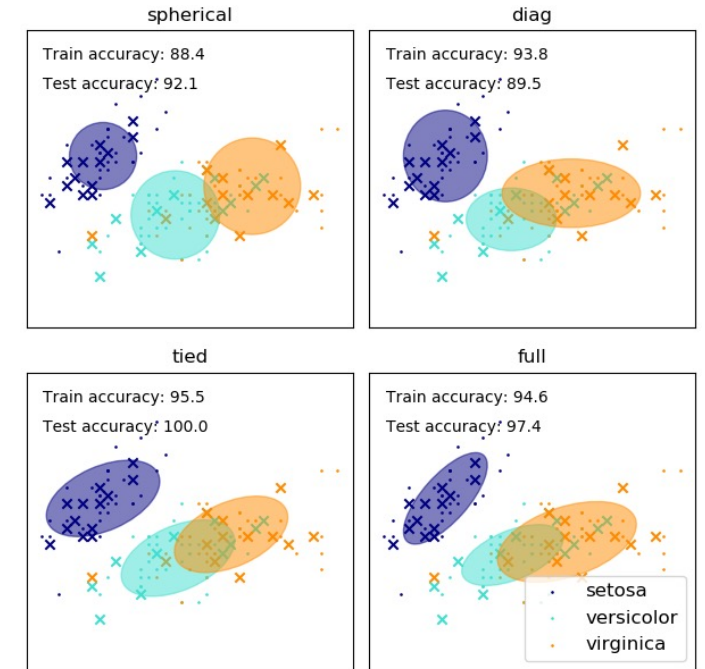
Clustering



K-means



Hierarchical clustering



Gaussian mixture models

K-means - Figure adapted from: <https://databricks.com/blog/2015/01/28/introducing-streaming-k-means-in-spark-1-2.html>

Hierarchical Clustering - Figure adapted from: <http://www.sthda.com/english/articles/28-hierarchical-clustering->

Mixture of Gaussians - Figure adapted from: <http://www.sthda.com/english/articles/28-hierarchical-clustering->

Inference vs Prediction – Linear Model $Y = \beta X + \epsilon$

statistical inference

Goal:

- Identify significant contributing variables (statistical null-hypothesis testing, p -values)

Uses:

- Scientific discovery. Ideal to uncover characteristics or true properties of the biological processes of the studied phenomenon.
- Useful to judge the individual relevance of each quantitative measure in impacting the response of interest.

pattern recognition

Goal:

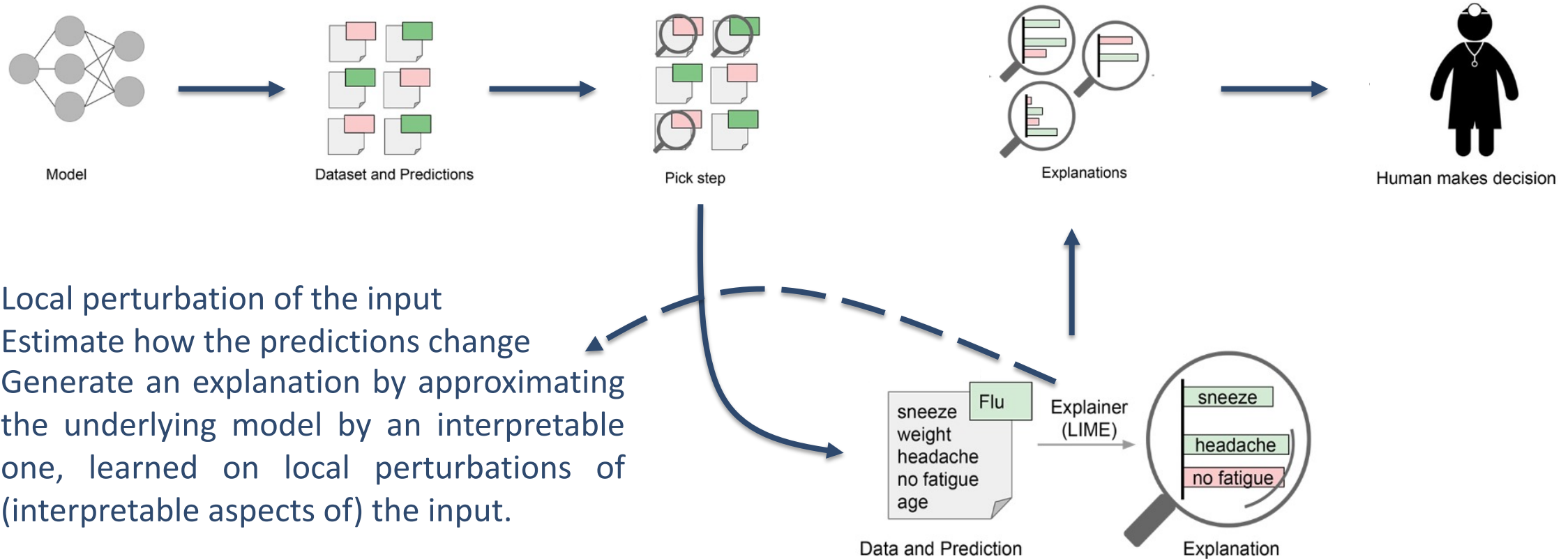
- Identify most predictive variable sets (out-of-sample prediction performance)

Uses:

- Pragmatic forecasting of biological processes.
- Tends to concern less regarding the data-generating process.

Diagnosing Features (\approx Interpretability)

Local Interpretable Model-Agnostic Explanations [*LIME*]



- Local perturbation of the input
- Estimate how the predictions change
- Generate an explanation by approximating the underlying model by an interpretable one, learned on local perturbations of (interpretable aspects of) the input.