Machine-Learning

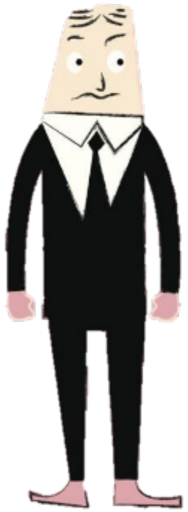
An introduction

by: Estefany Suárez & Jake Vogel

29/06/2021

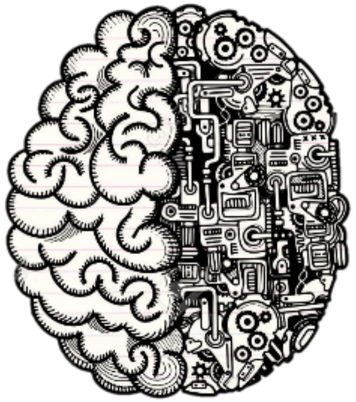
Machine Learning (ML)

Learn from experience

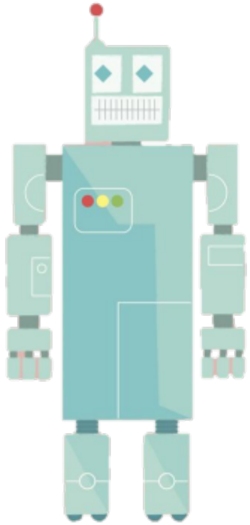


DATA

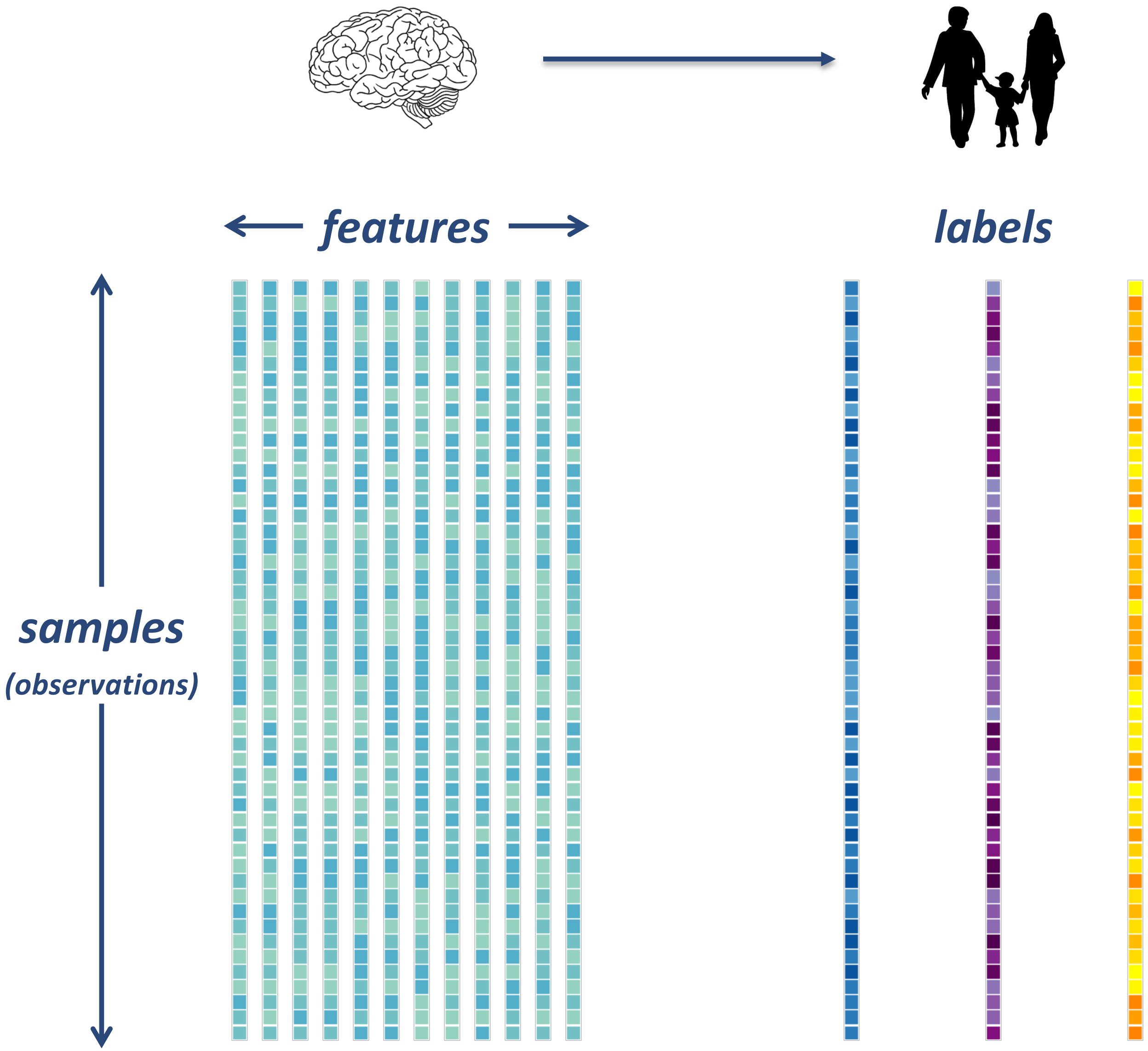
Learn from ~~experience~~



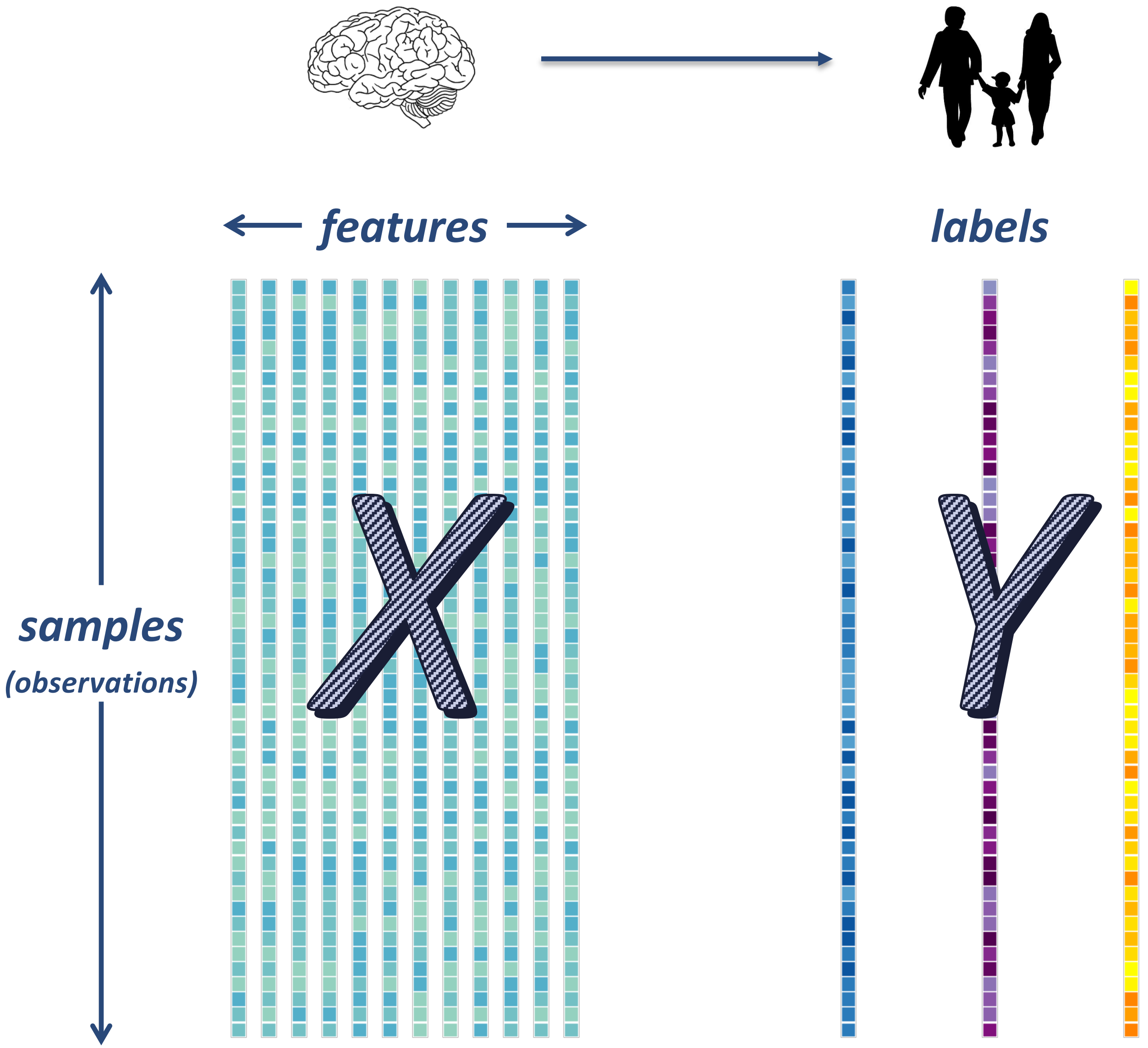
Follow instructions



Terminology



Terminology



ML algorithms

learn a function to map *X* to *Y*
prediction

Supervised Learning

$$x \text{ , } y$$

$$f: x \rightarrow y$$

$$y = f(x)$$

Semi-supervised Learning

create useful representations of data X

Unsupervised Learning

$$x \text{ , } \cancel{y}$$

ML algorithms

learn a function to map X to Y
prediction

Supervised Learning

Discrete Y — Classification

Continuous Y — Regression

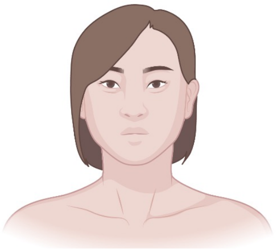
create useful representations of data X

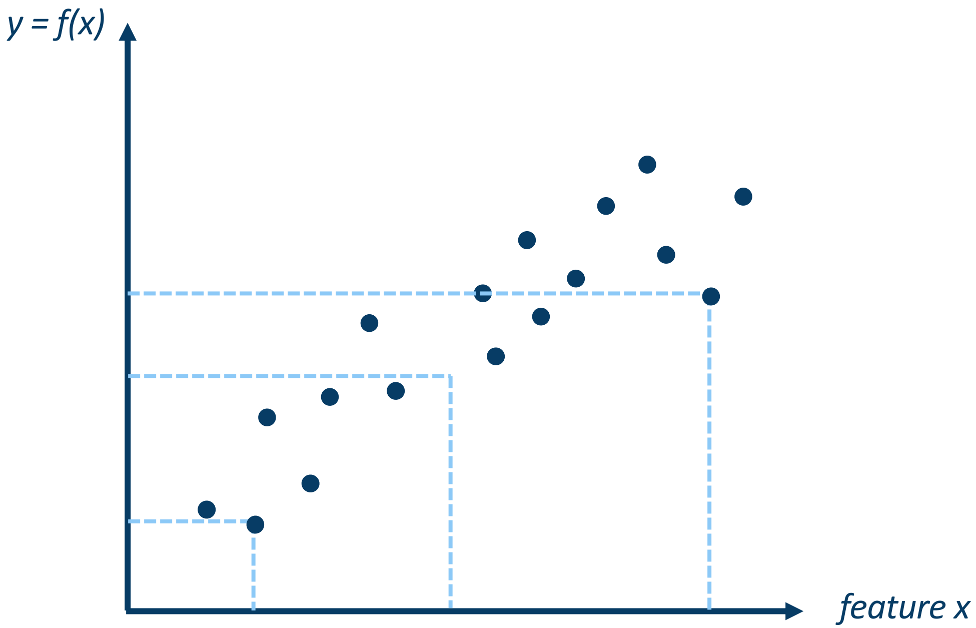
Unsupervised Learning

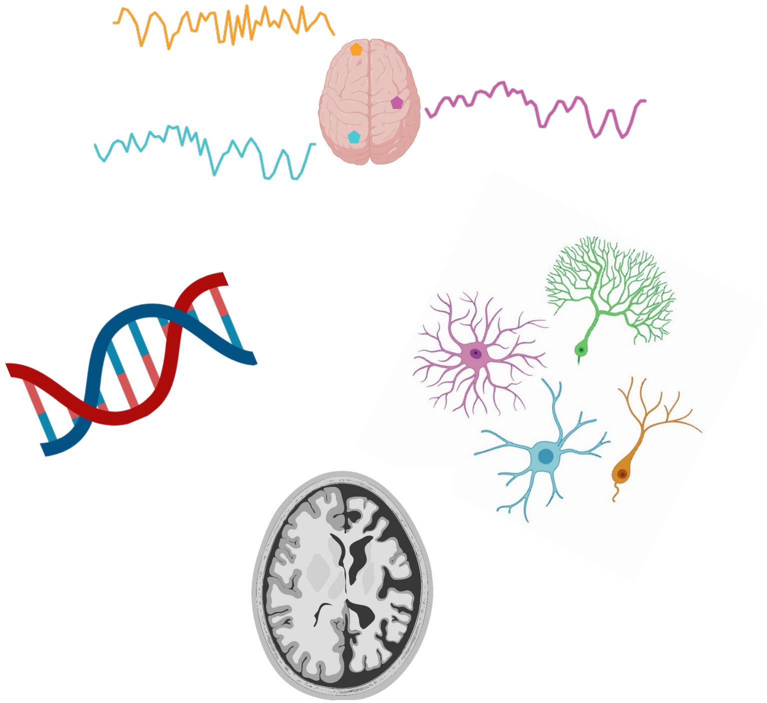
Clustering

Dimensionality Reduction

Regression





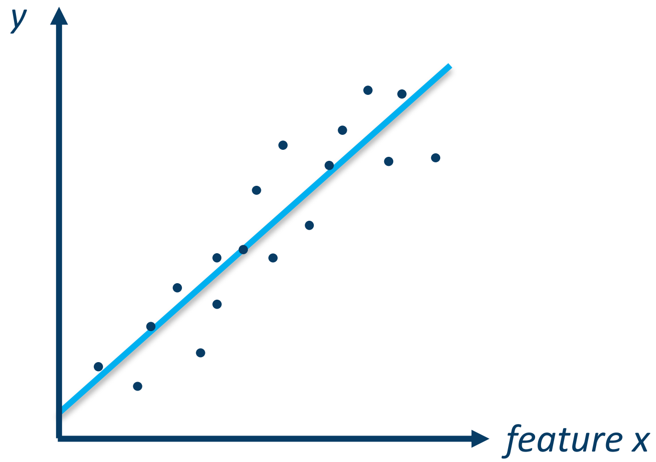


Regression

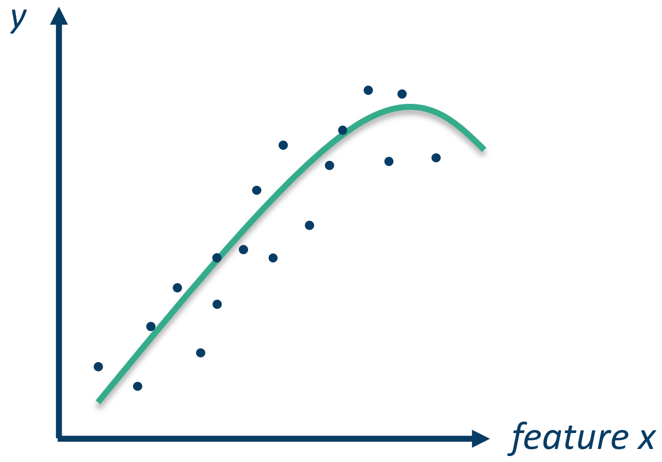
$$f: x \to y$$

$$y = f(x)$$

$$y = \boldsymbol{a}x + \boldsymbol{b}$$



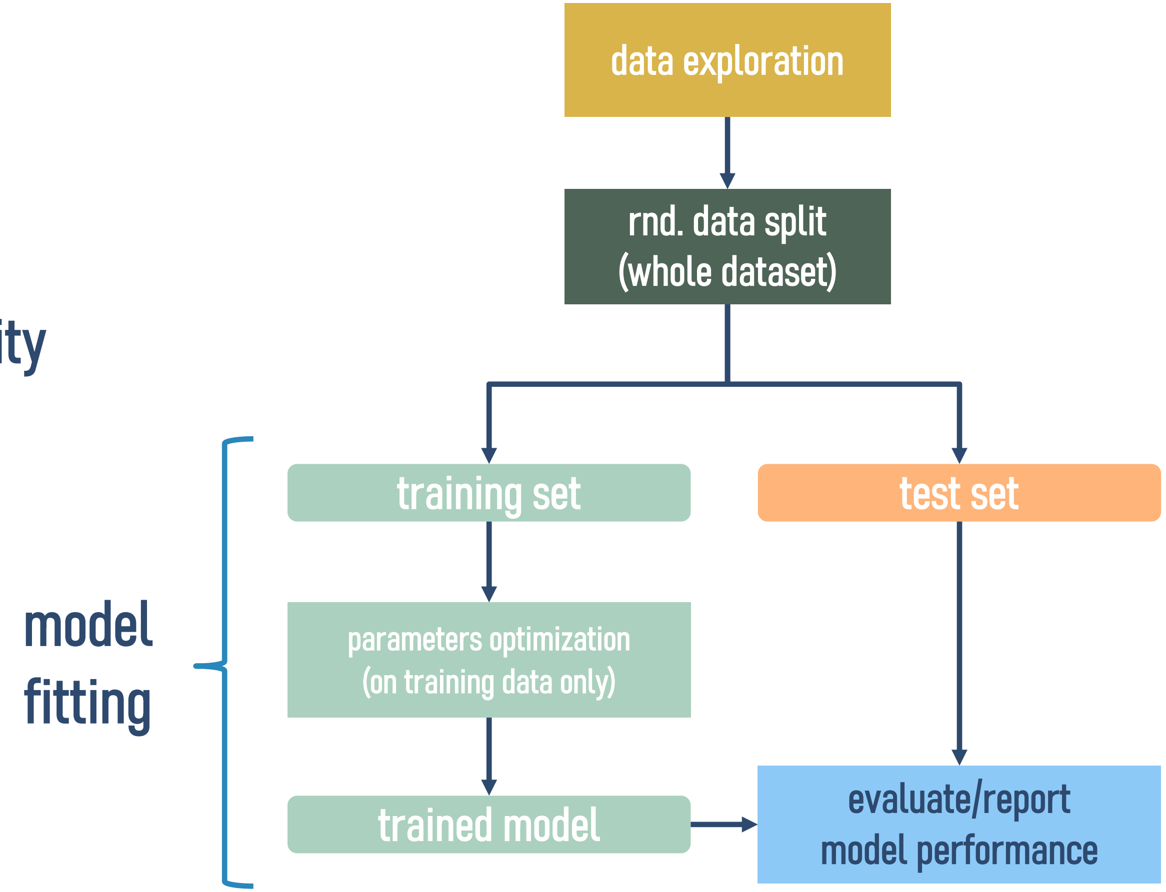
$$y = \boldsymbol{a}x^2 + \boldsymbol{b}x + \boldsymbol{c}$$



Typical ML Pipeline

Why cross-validation (CV)?

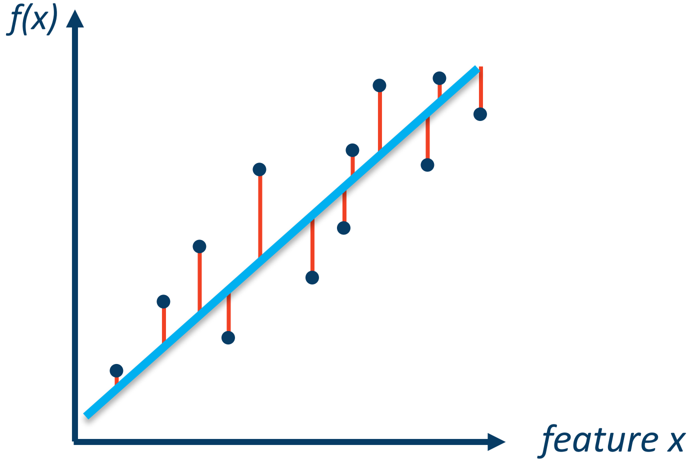
- Avoid overfitting
- Improve model generalizability



ML basic ingredients

$f: x \rightarrow y$

$y = f(x)$

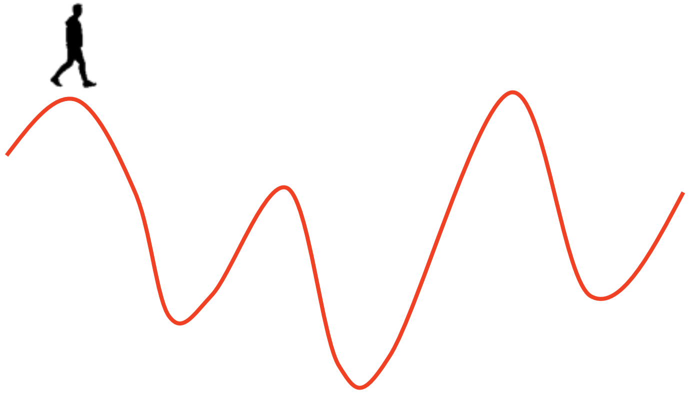


0

1 Model or estimator ($f: x \rightarrow y$) $\longrightarrow$ $y = ax + b$

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

3 Optimization algorithm $\longrightarrow$ gradient descent

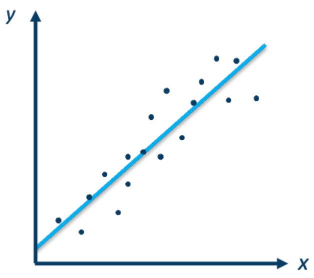


ML basic ingredients

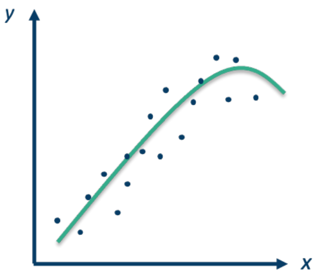
① Model *(estimator)* selection

linear regression

$y = ax + b$

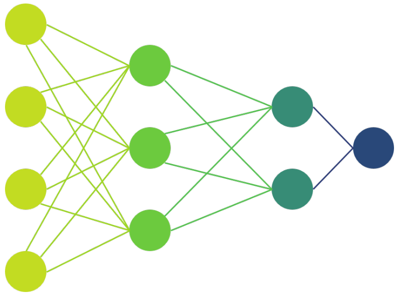


$y = ax^2 + bx + c$



polynomial regression

multi-layer perceptron (ANN)



② Loss *(cost)* 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. $= 1 - \frac{\mathrm{Var}(y - \hat{y})}{\mathrm{Var}(y)}$

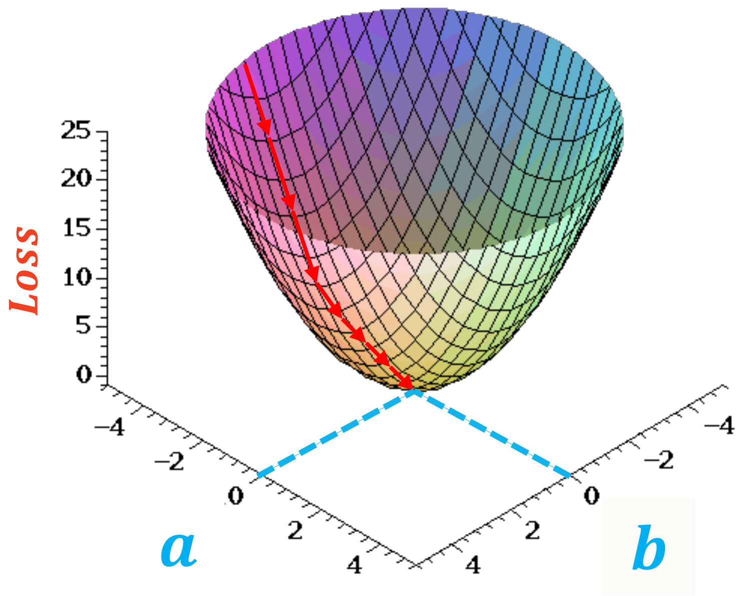
$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})}$

ML basic ingredients

③ Optimization algorithm for model parameters!

$$y = \boldsymbol{a}x + \boldsymbol{b}$$

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



Gradient Descent

Training data

x_{train} y_{train}

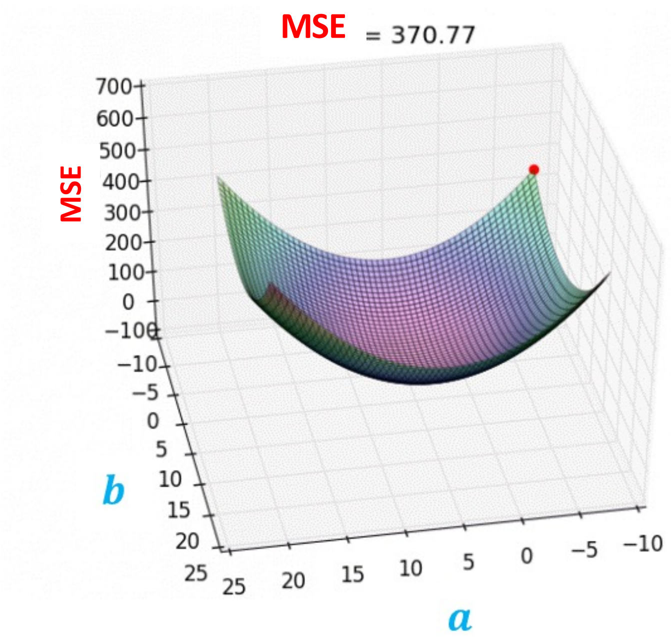
Initialize model parameters ($\boldsymbol{a}$,$\boldsymbol{b}$) randomly iterate between:

1) Estimate output $\hat{\boldsymbol{y}}_{\boldsymbol{i}} = \boldsymbol{a}x_i + \boldsymbol{b}$

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

3) Compute the gradient

4) Update parameters ($\boldsymbol{a}$,$\boldsymbol{b}$)



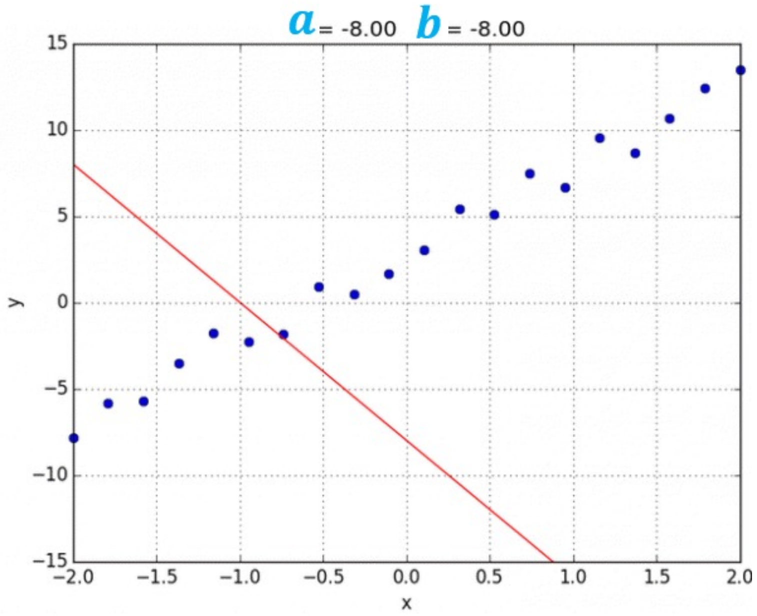
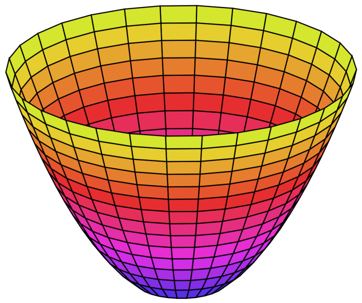


Figure adapted from: https://giphy.com/gifs/gradient-O9rcZVmRcEGqI

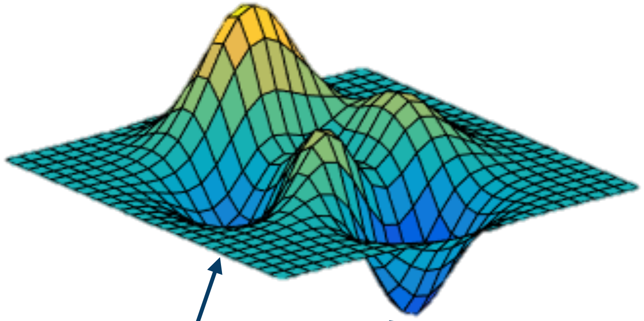
There is a problem though …

simple loss function



global
minimum

convoluted loss function



multiple local
minima

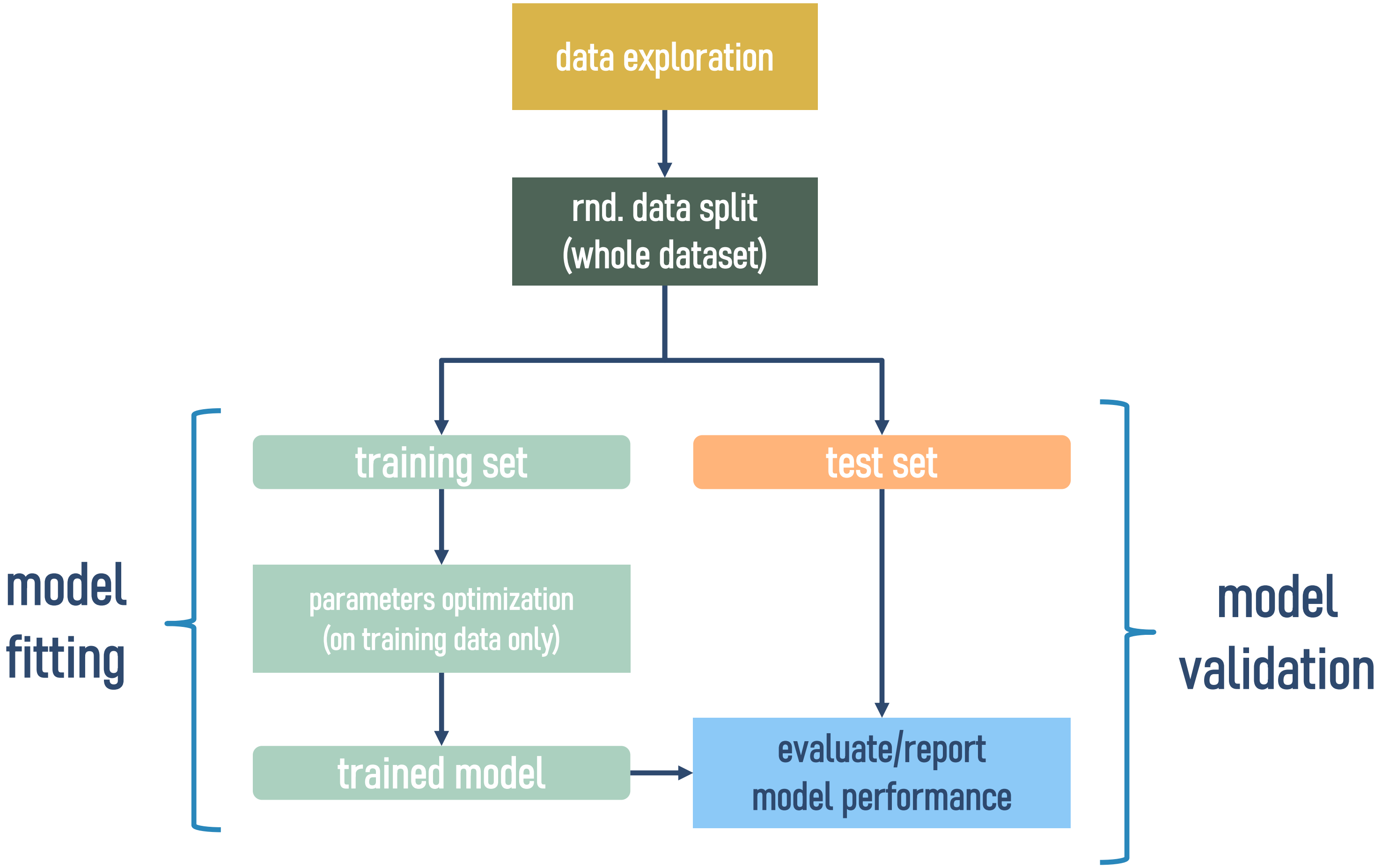
*

Re-use your data . . .

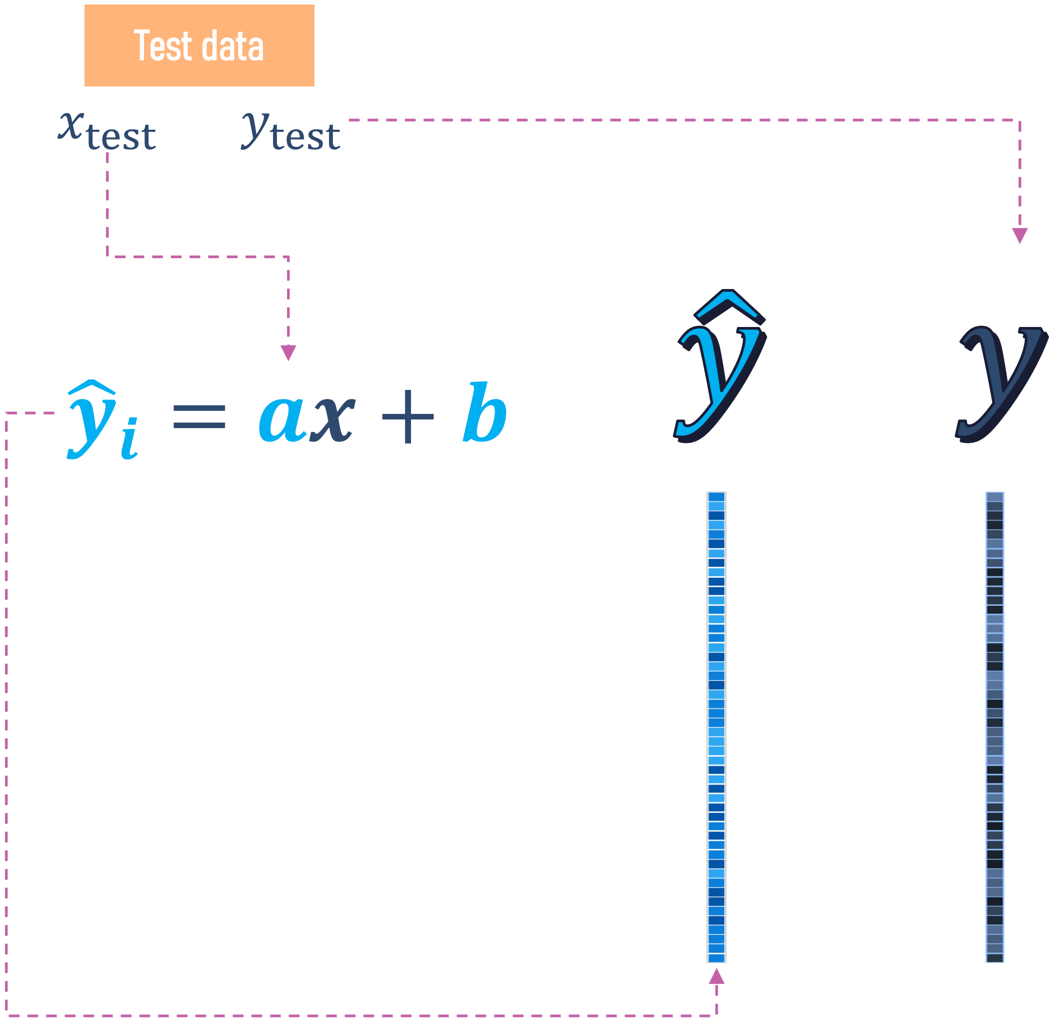
multiple training epochs !

*Figure taken from: http://www.duncanmalcolm.com/startup-data-analytics-metric-tools/

Typical ML Pipeline



Model Validation



score function

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

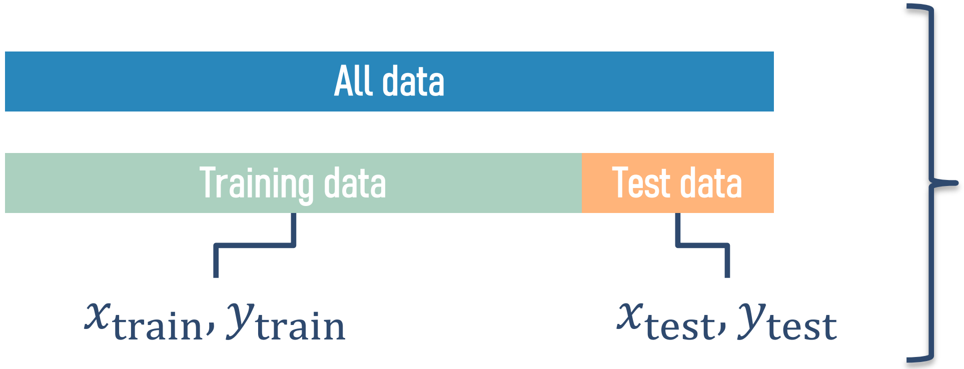
$$\text{mean absolute error} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

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

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

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

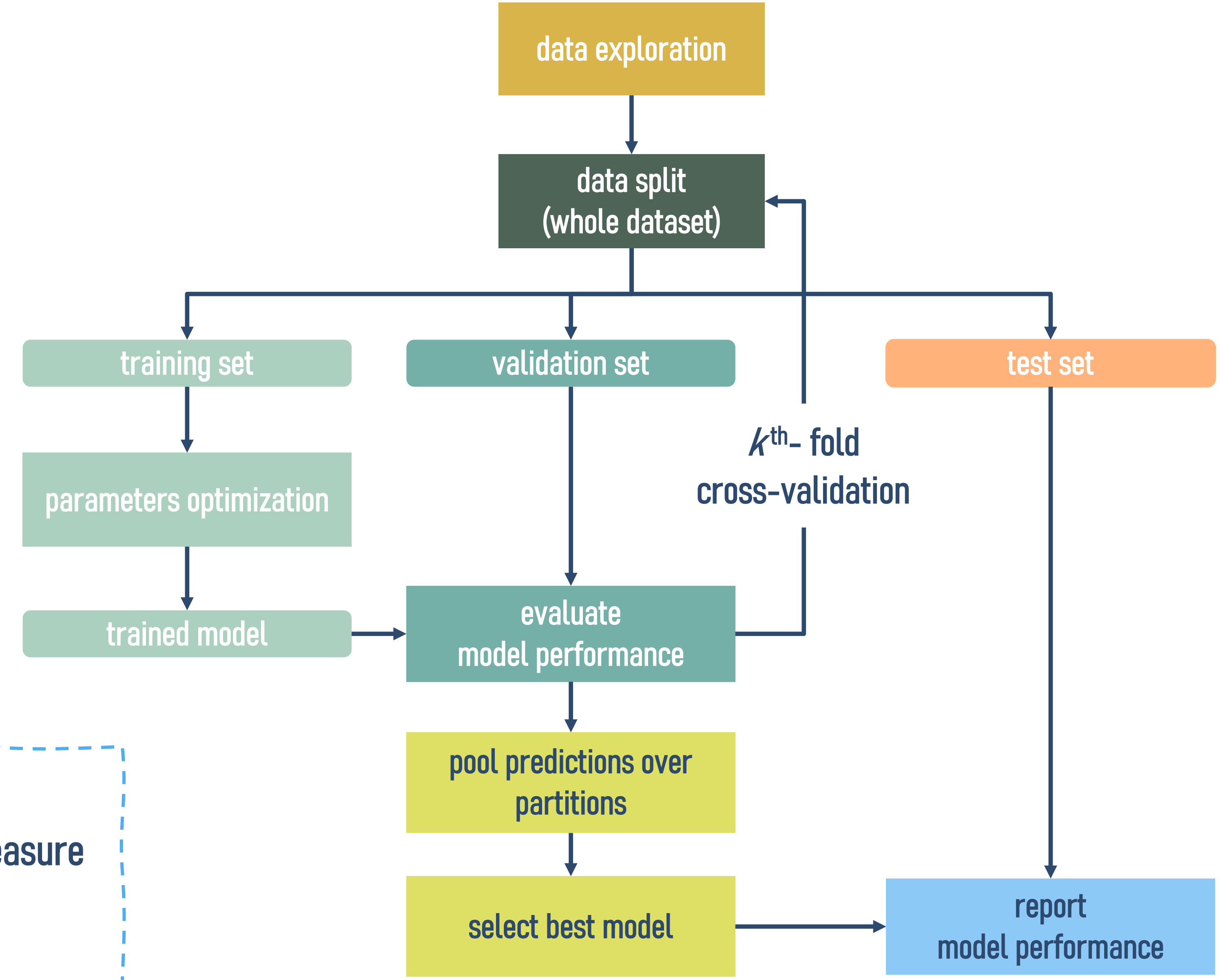
We have yet another problem …



BUT … What if your outlier samples/observations happen to be in the test set?

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

Typical ML Pipeline



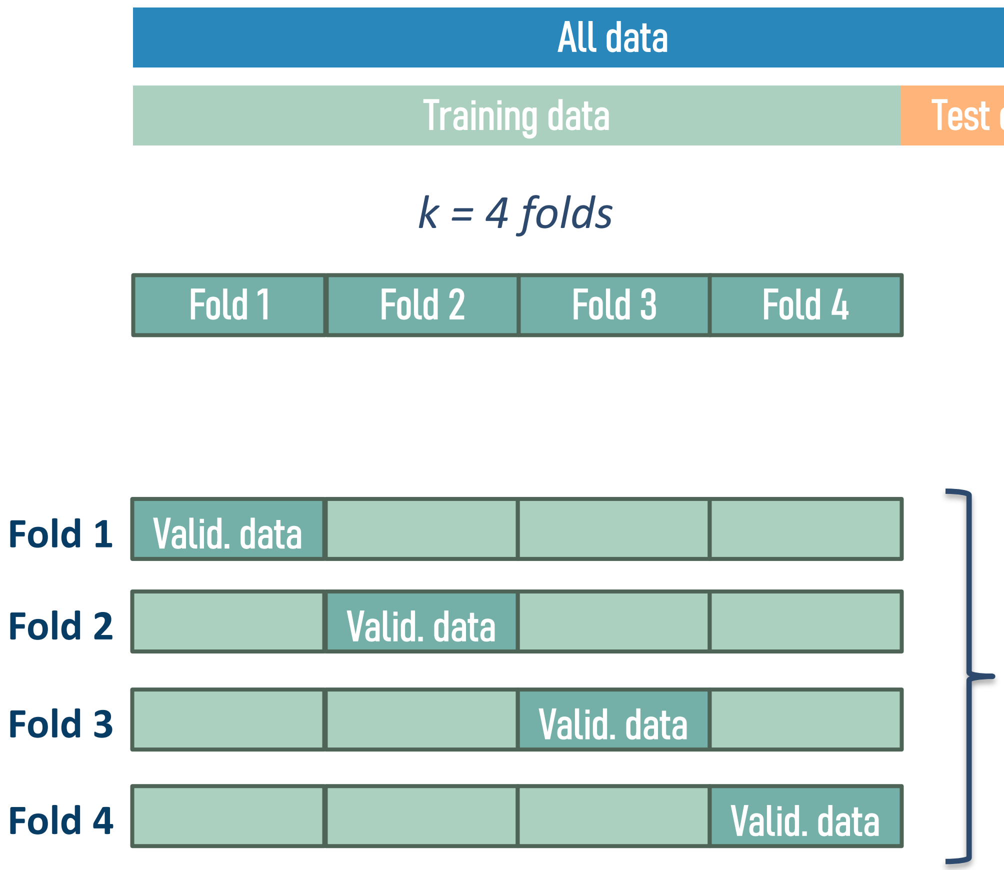
Why k^{th}-fold CV?

- Provide a more accurate performance measure

  [average, standard deviation, range]

- Model selection
- *Hyper*-parameter tuning

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

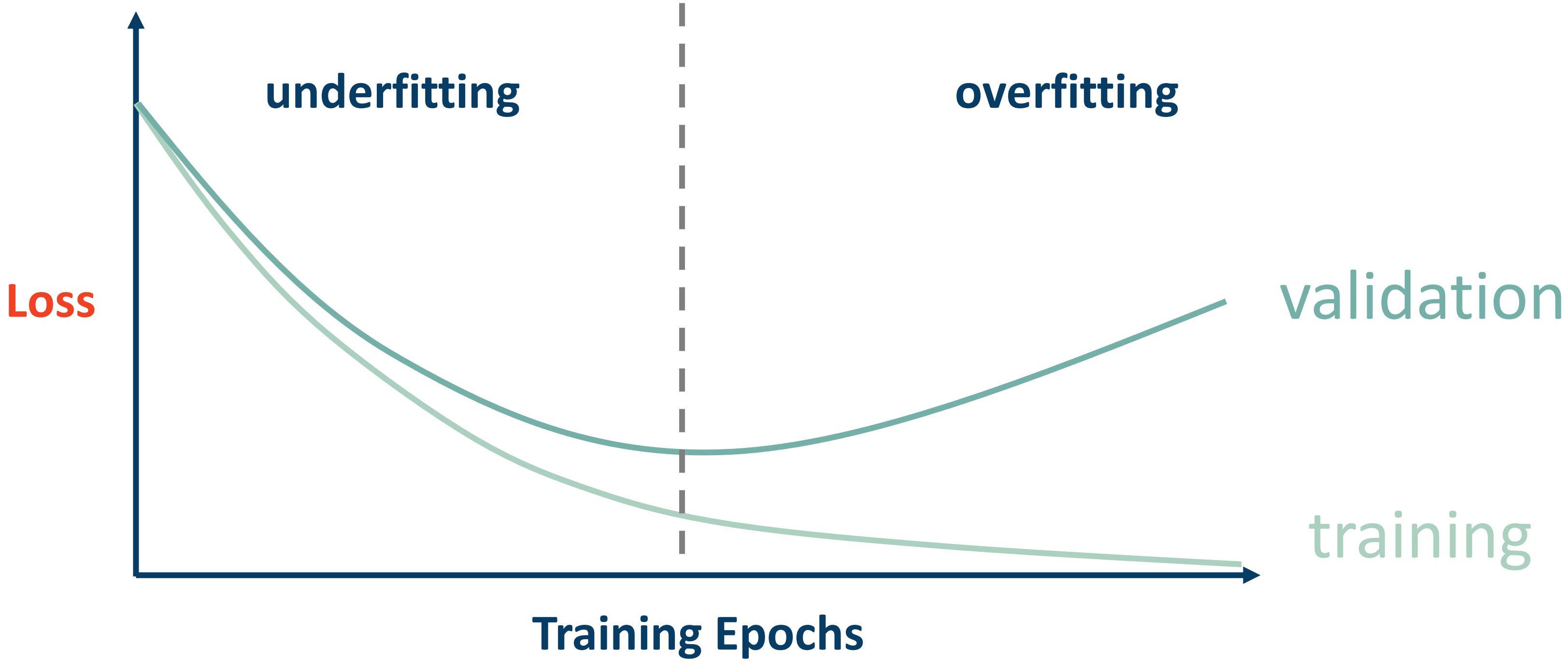


1. For each fold:
   I. Train model parameters with <u>training</u> set
   II. Evaluate training with <u>validation</u> set
2. Based on <u>validation</u> set, select *optimal* model and hyper-parameters
3. Only at the very, very **END**, report error on <u>test</u> set

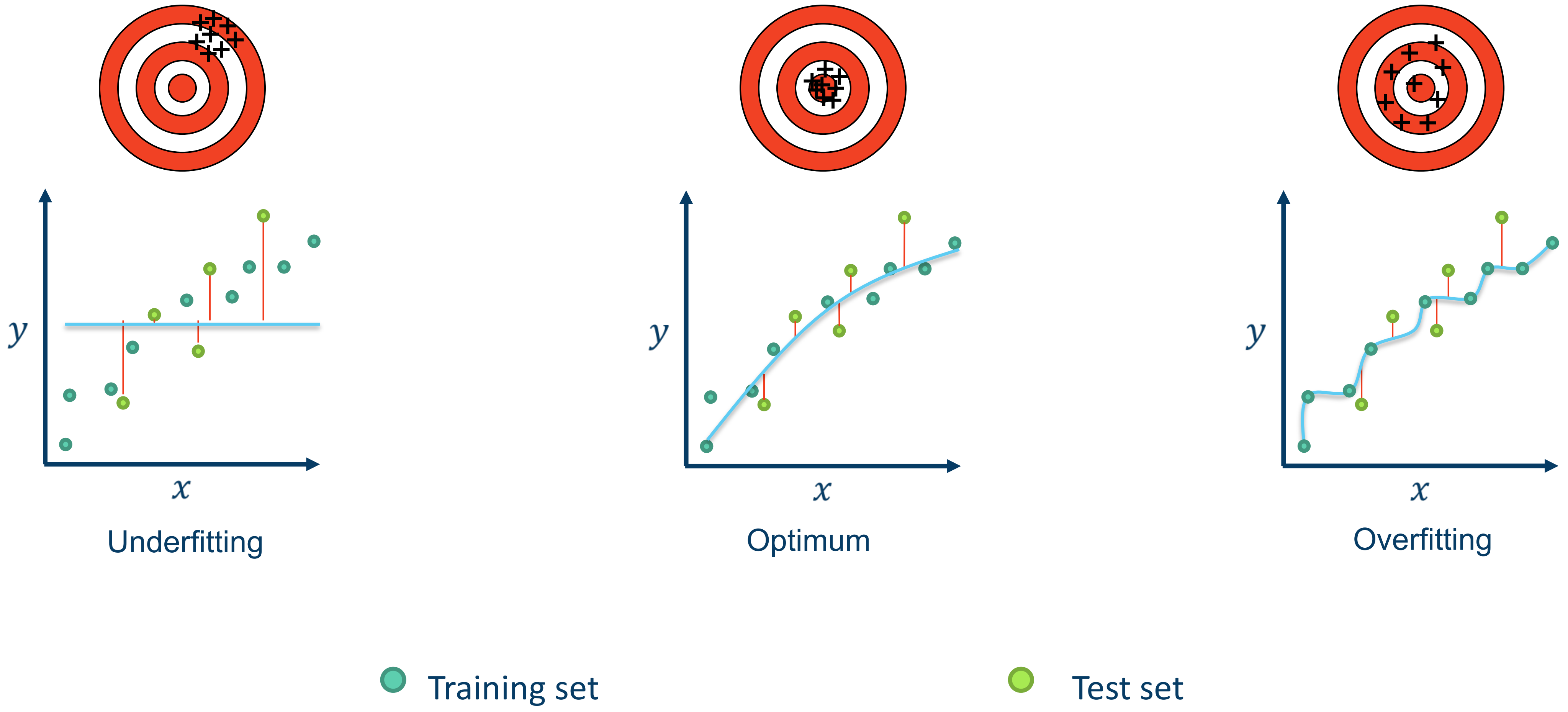
Overfitting

k^{th}-fold

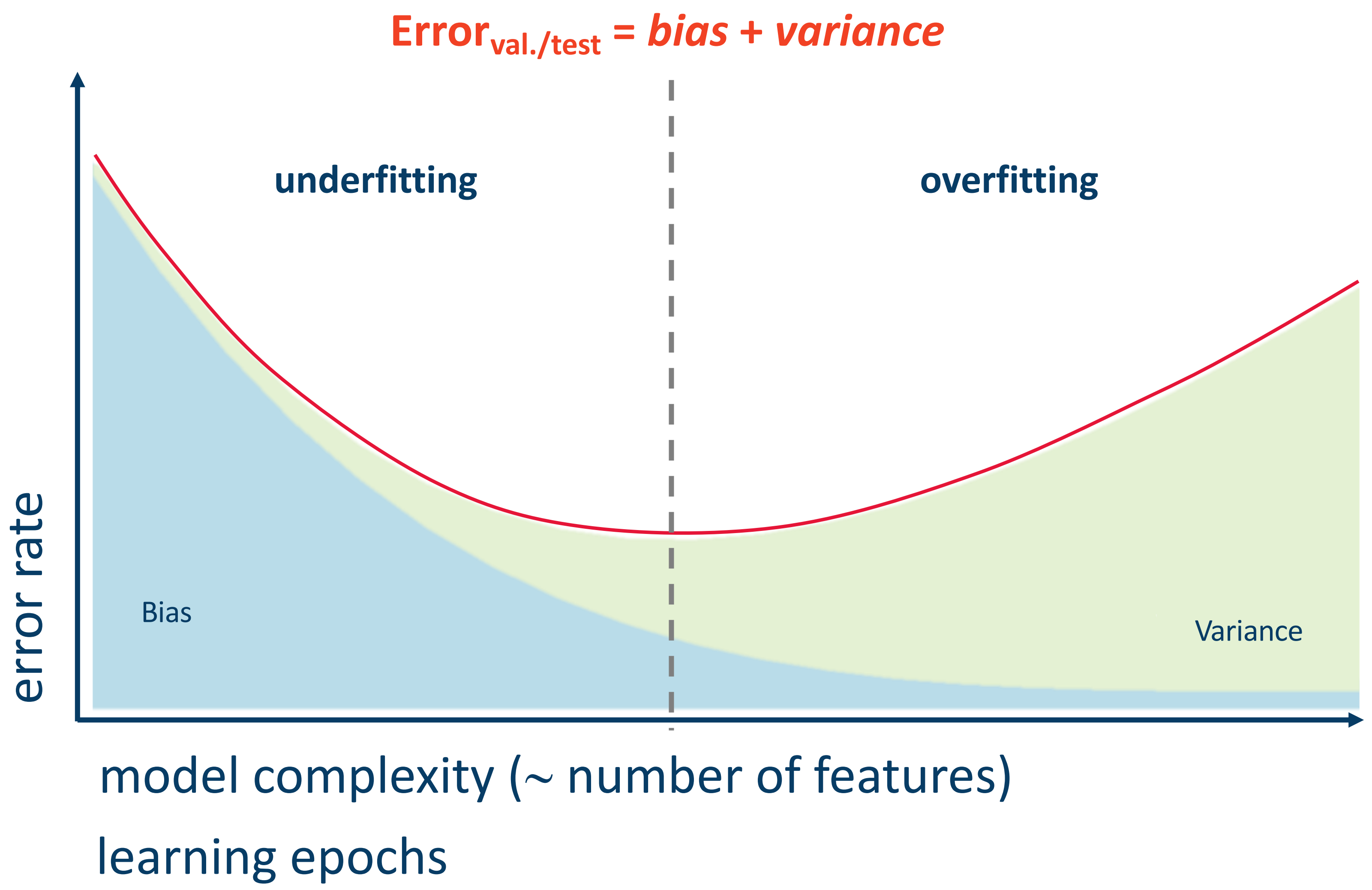
How many TE?



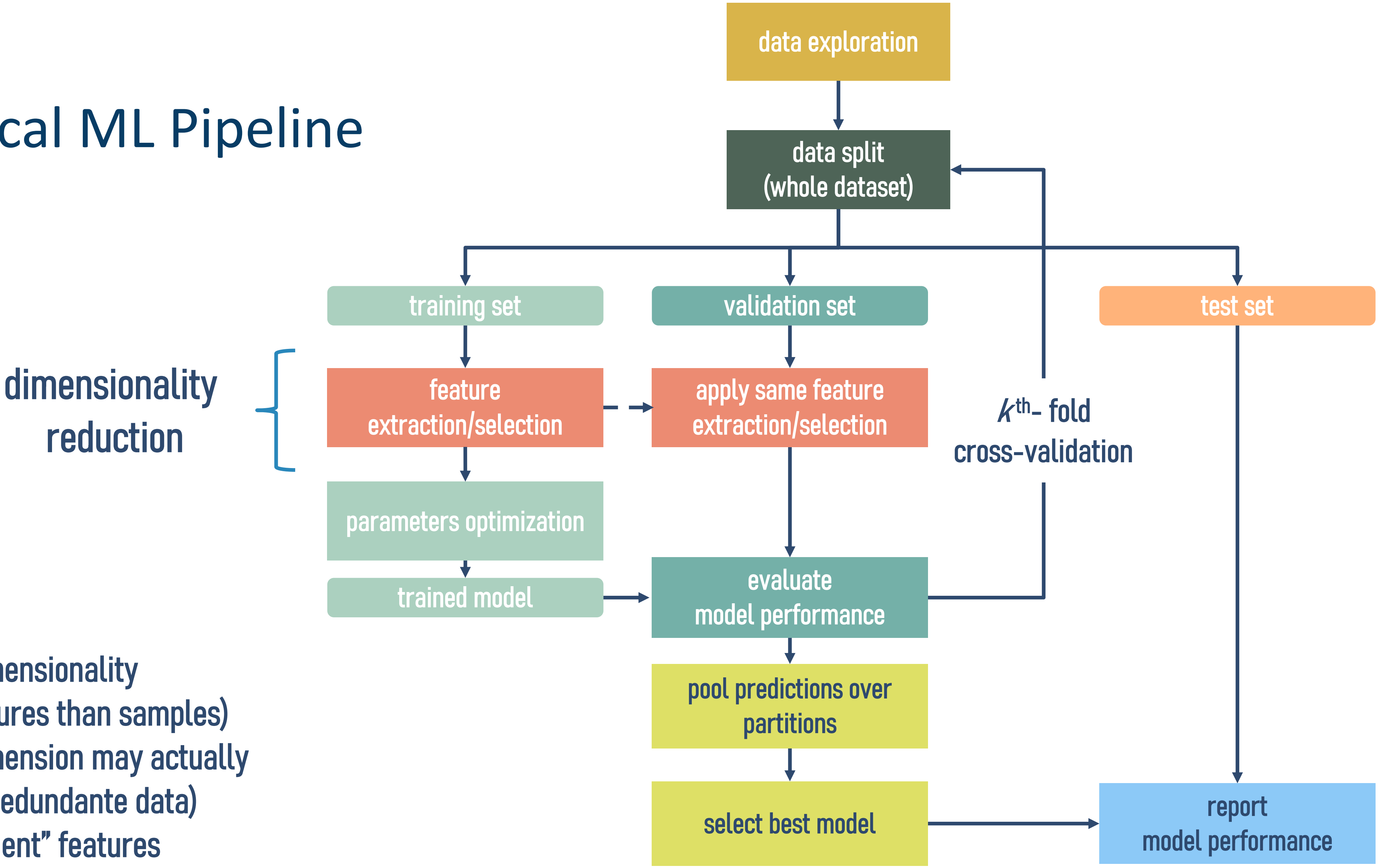
Bias-Variance Trade-Off



Bias-Variance Trade-Off

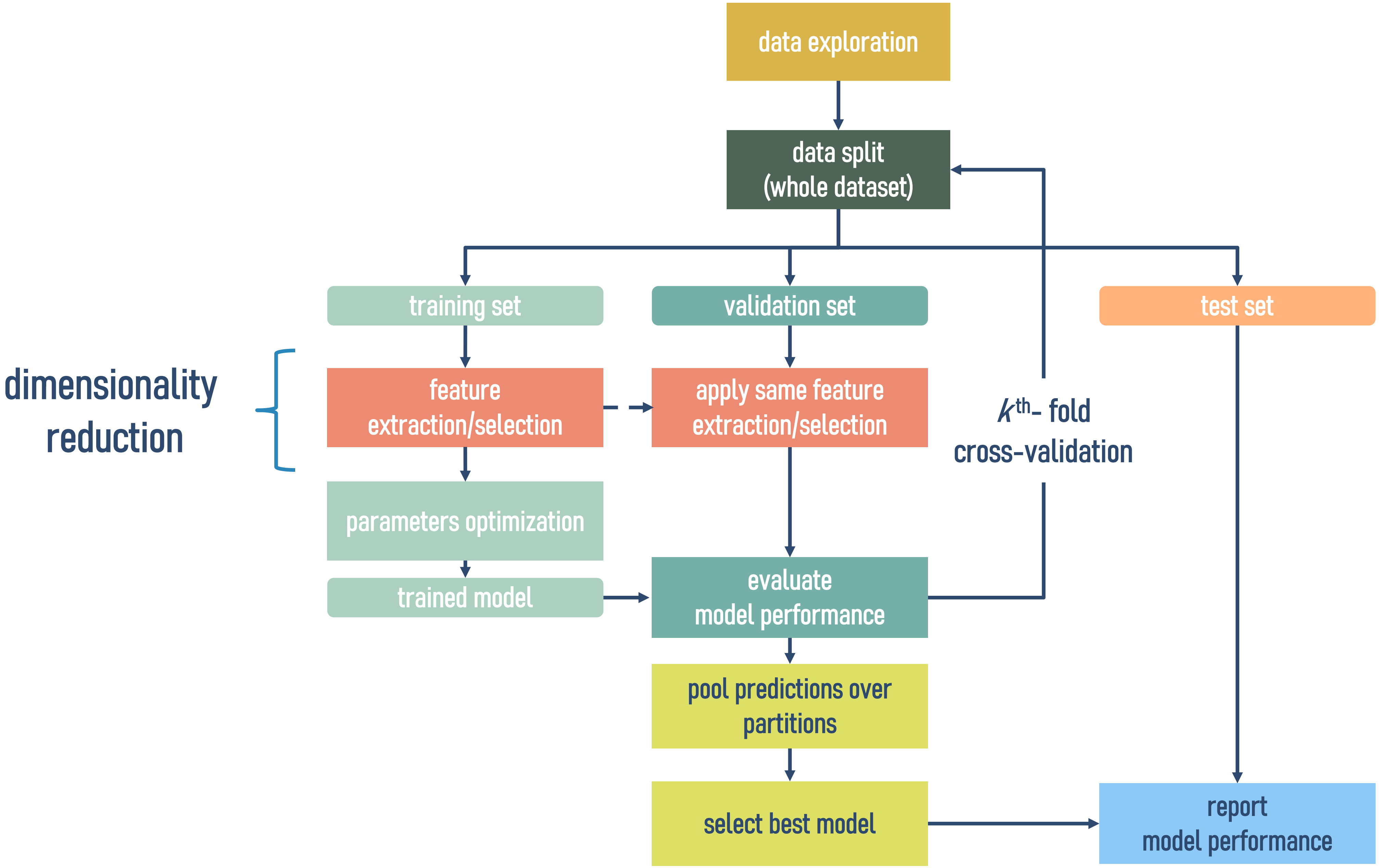


Typical ML Pipeline

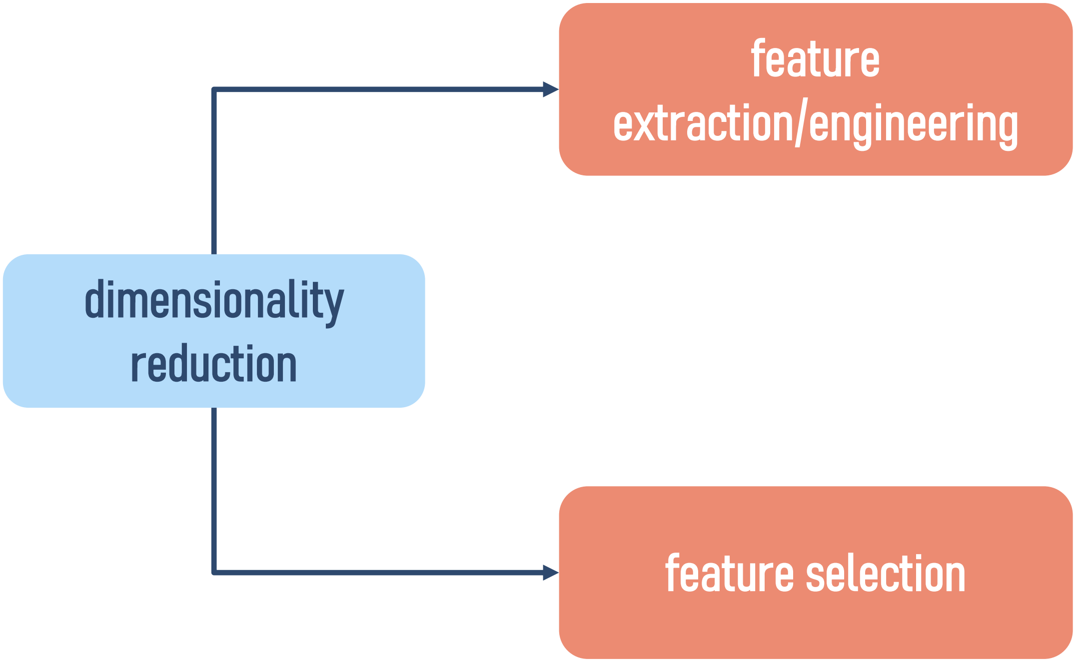


WHY DR ?

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


data exploration
data split
(whole dataset)
training set
validation set
test set
dimensionality
reduction
feature
extraction/selection
apply same feature
extraction/selection
k^{th}- fold
cross-validation
parameters optimization
trained model
evaluate
model performance
pool predictions over
partitions
select best model
report
model performance

Tip 1: Dimensionality Reduction



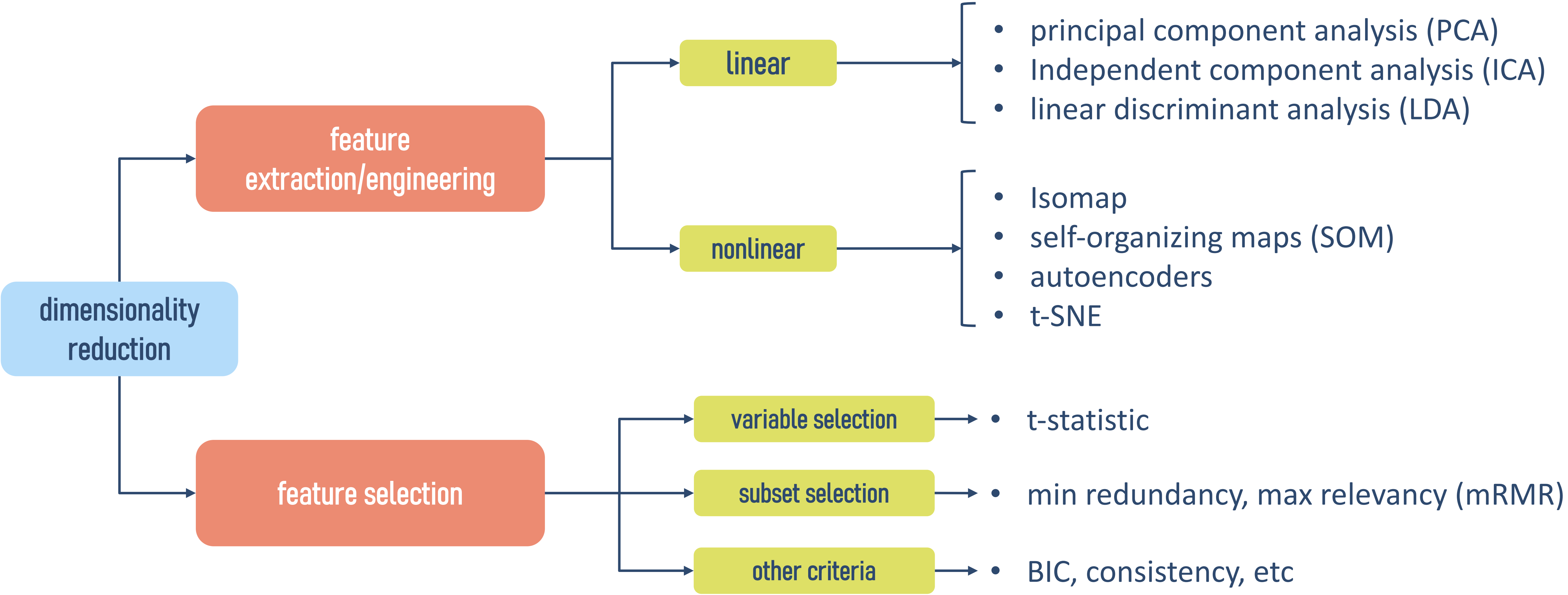
- Compact representation of the data
- Maps input features into a lower dimensional space

e.g. If features are $X = [x_1, x_2, x_3, \mathrm{x}_4]$
then $\mathrm{Z} = T(X) = [c_1 x_1 + c_2 x_2, \quad x_3 * x_4]$

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

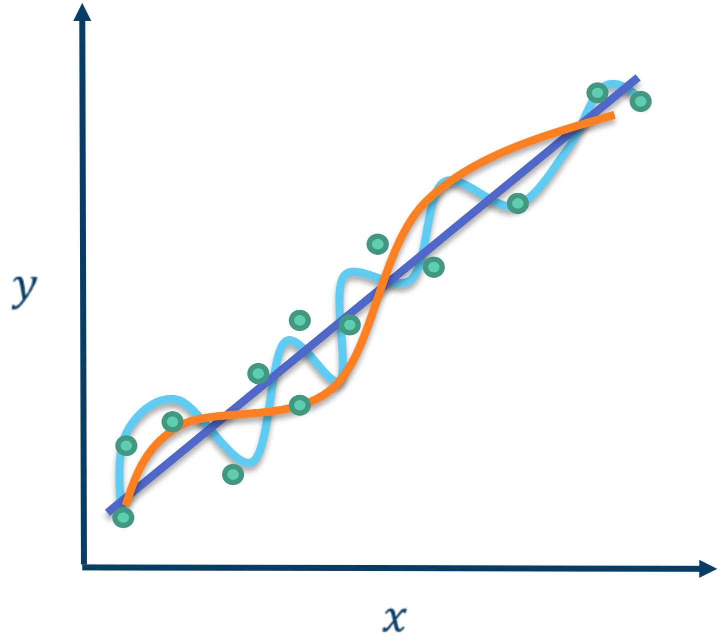
e.g. $\mathrm{Z} = S(X) = [x_2, \; x_3]$

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**

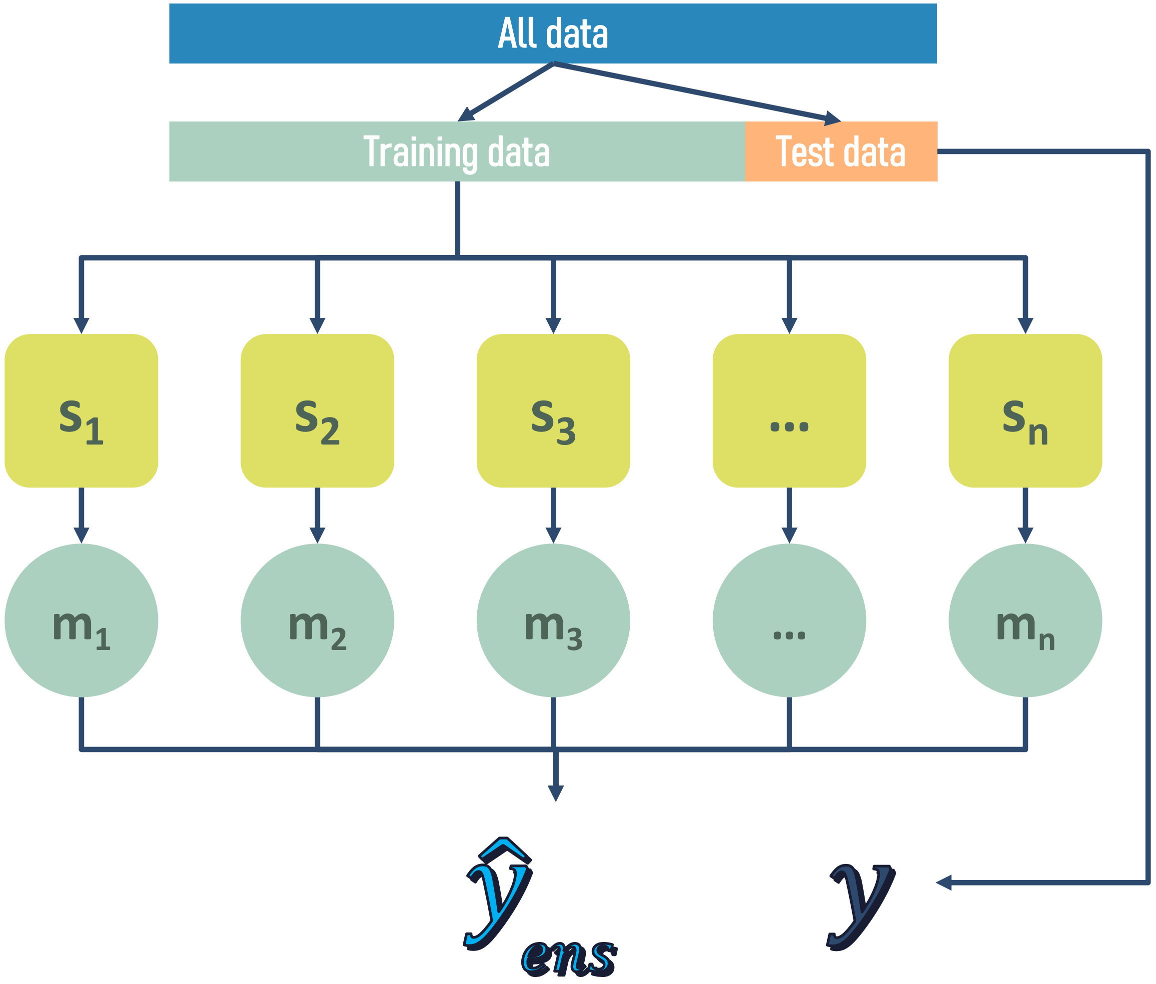
$$MSE = \sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

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

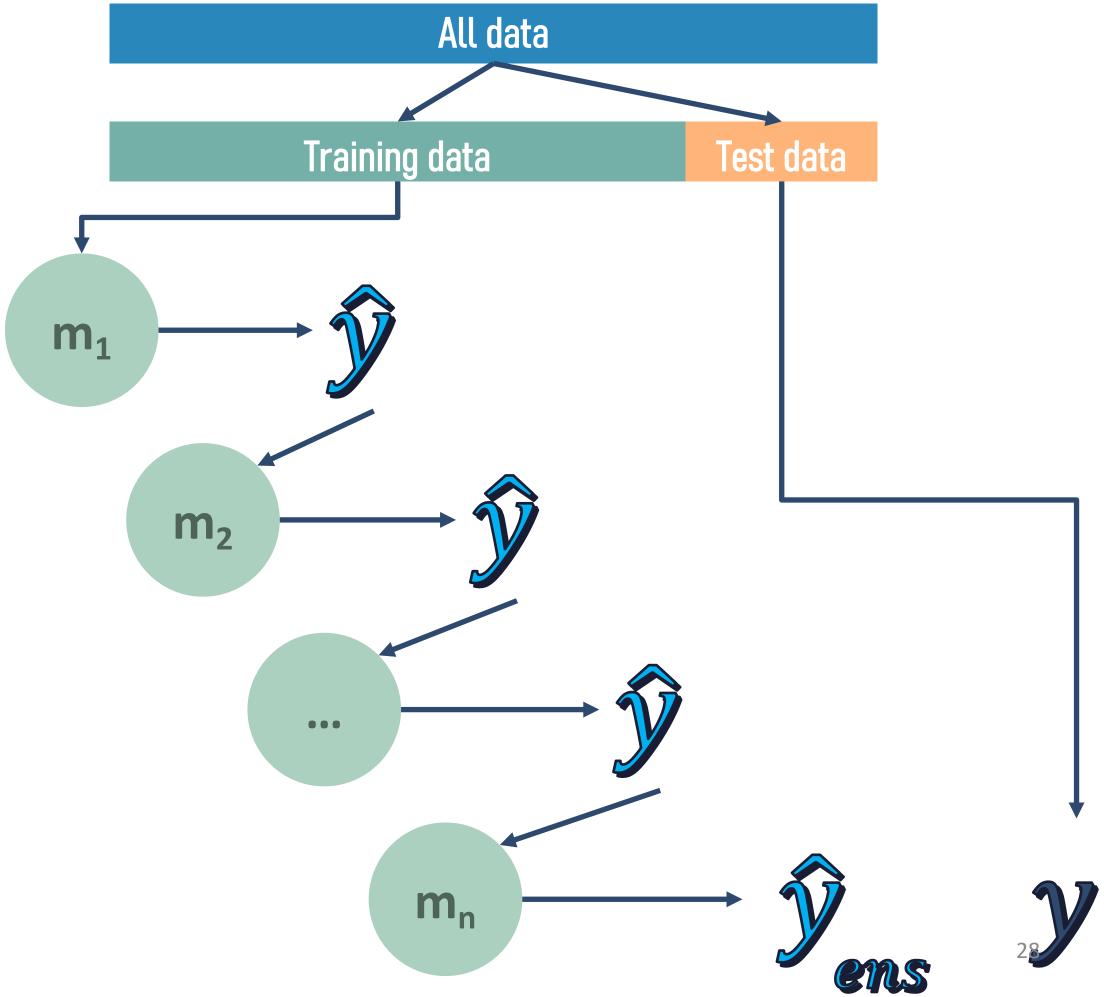
$$MSE = \sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda \sum_{j=1}^{p} {\beta_j}^2$$

Tip 3: Ensemble Methods

bagging = bootstrapp resampling + aggregation



boosting



Sources of Bias

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

Pradeep Reddy Raamana (slide 63 - https://drive.google.com/file/d/1jXWQT9P4CmJgBShjwqymKiI_jOH-bb7q/view)

ML algorithms

learn a function to map X to Y
prediction

create useful
representations of data X

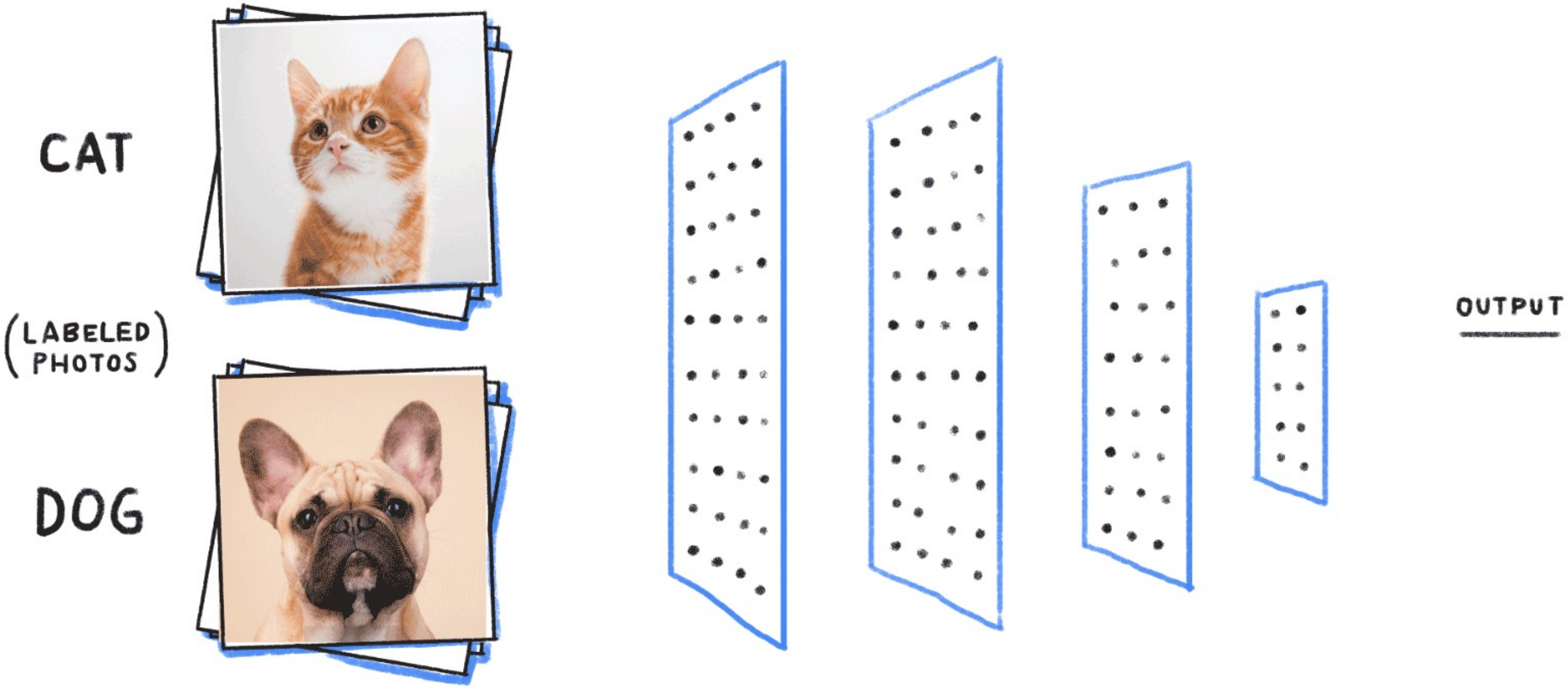
Supervised Learning

Unsupervised Learning

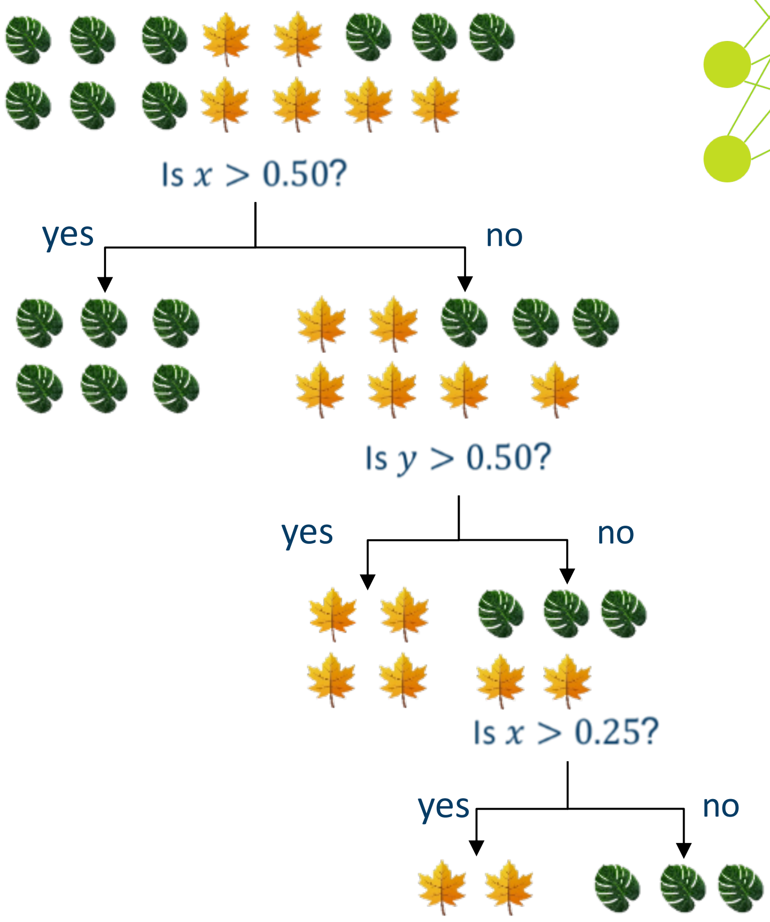
Discrete Y — Classification — Clustering

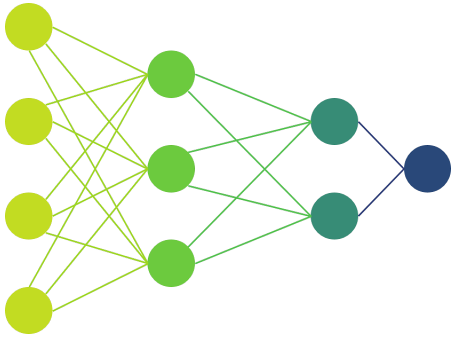
Continuous Y — Regression — Dimensionality Reduction

Classification



Classification

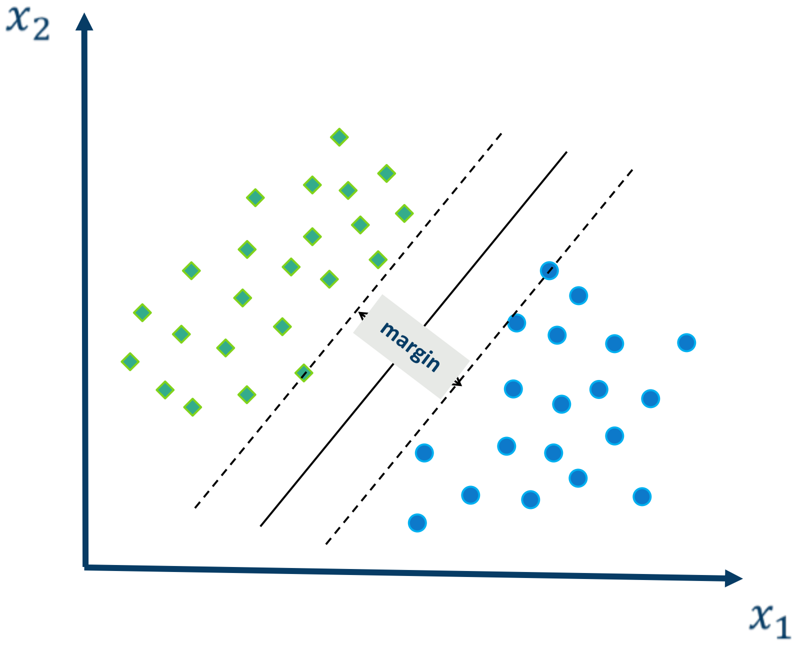




- Support Vector Machine (SVM)

- 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

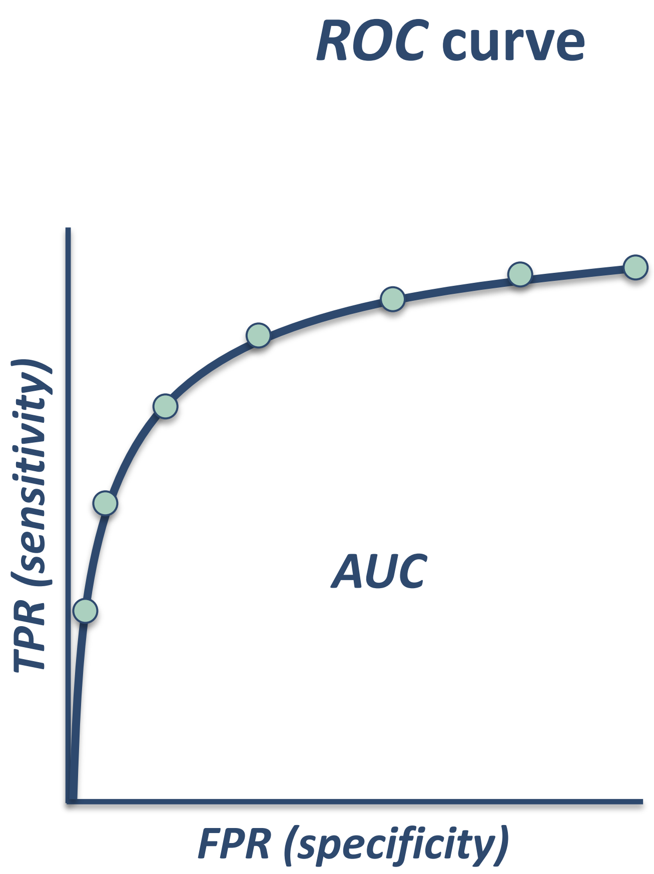
Sensitivity (or recall) $TPR = \frac{T_p}{P} = \frac{T_p}{T_p + F_n}$

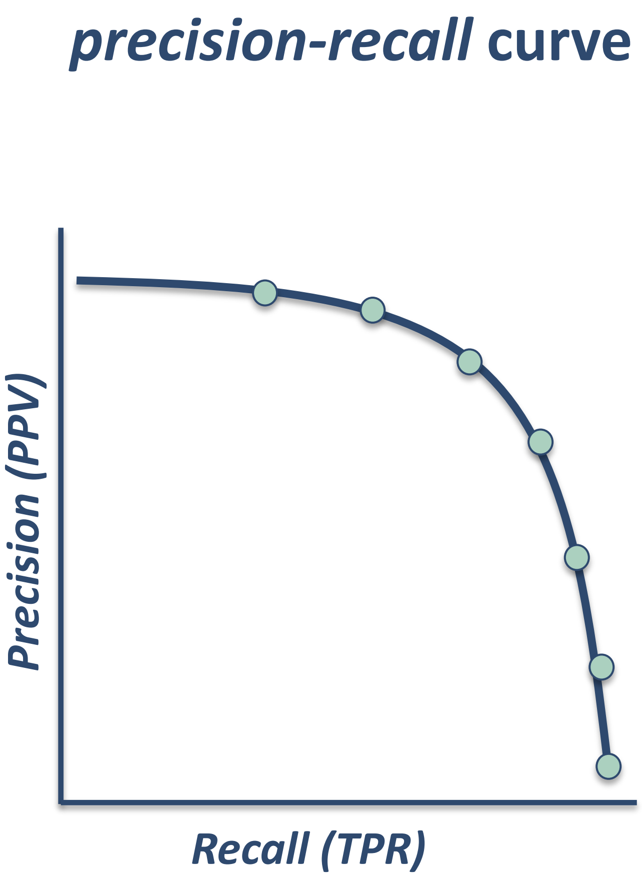
$FPR = \frac{F_p}{N} = \frac{F_p}{T_n + F_p}$ *Specificity (or selectivity)*

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

Performance Metrics – Binary Classification

(probabilistic classifiers)





Appropriate when classes are imbalanced!

ML algorithms

automatically find patterns in data - **prediction**

Supervised Learning

Discrete — Classification

Continuous — Regression

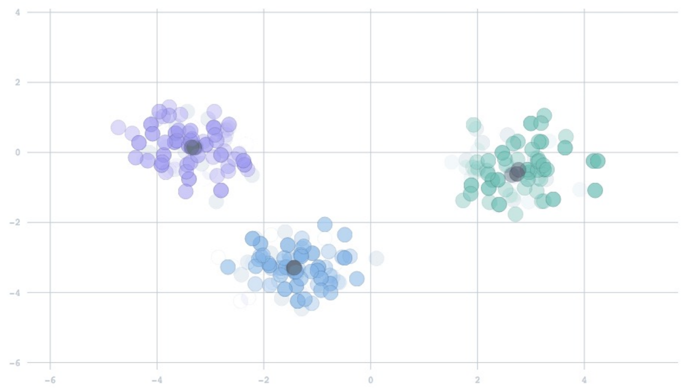
create useful representations of data

Unsupervised Learning

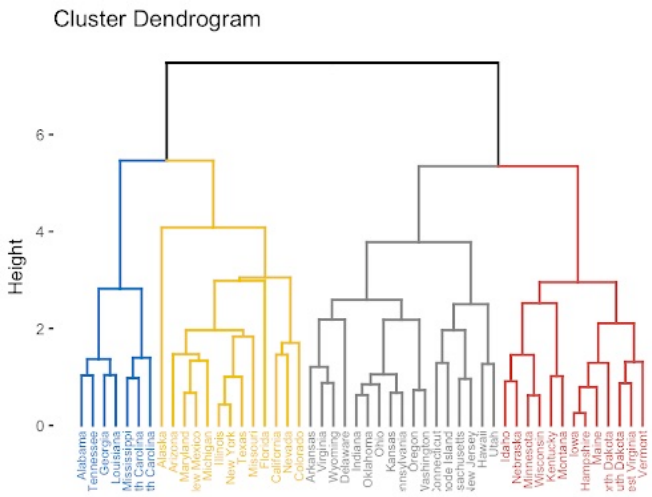
Clustering

Dimensionality Reduction

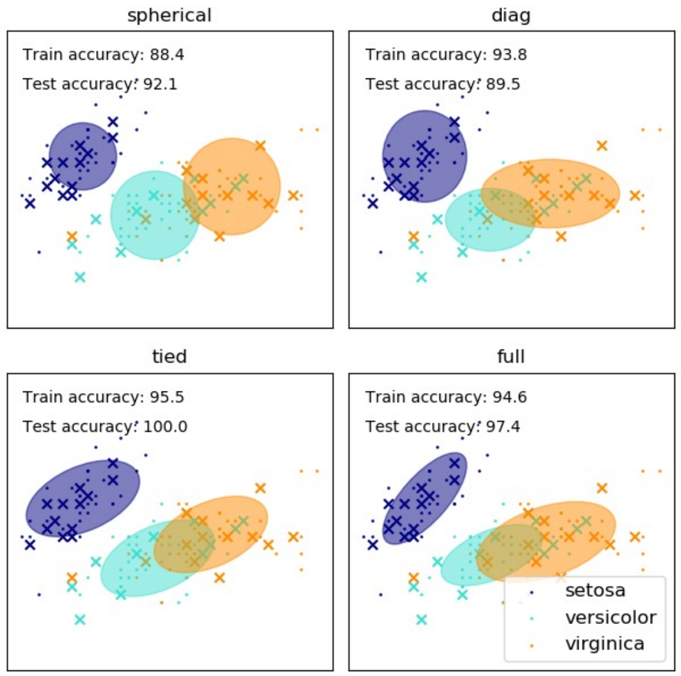
Clustering



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 $\mathbf{Y} = \boldsymbol{\beta X} + \boldsymbol{\epsilon}$

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)

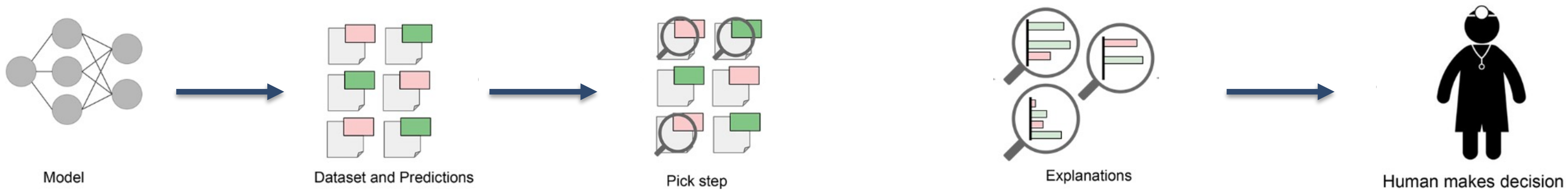
Uses:

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

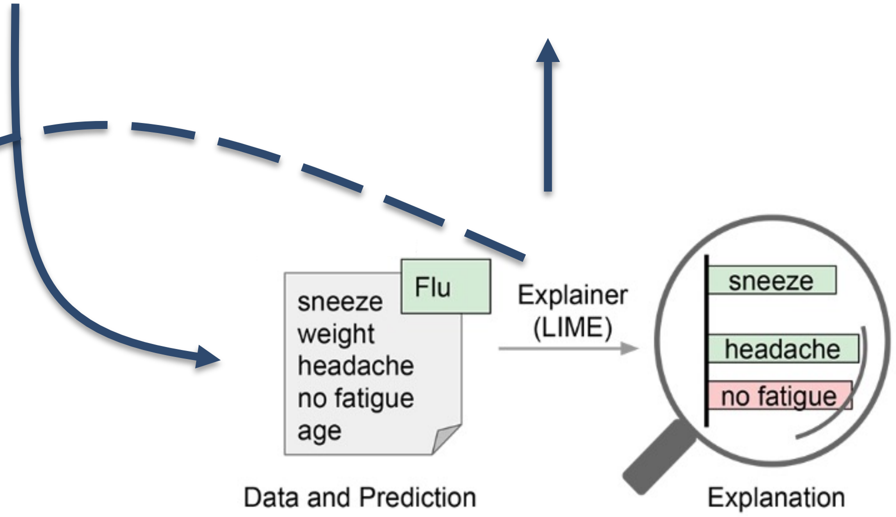
(Bzdok, et al., 2018, *bioRχiv)*

Diagnosing Features (≈ 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.



(Tulio-Ribeiro, et al., 2016, *bioRχiv)*