

# **Machine Learning:**

# **Lecture #1**

Jennifer Ngadiuba (Fermilab)  
University of Pavia, May 8-12 2023

# Overview of the lectures

- **Day 1:**

- Introduction to Machine Learning fundamentals
- Linear Models

- **Day 2:**

- Neural Networks
- Deep Neural Networks
- Convolutional Neural Networks

- **Day 3:**

- Recurrent Neural Networks
- Graph Neural Networks (part 1)

- **Day 4:**

- Graph Neural Networks (part 2)
- Transformers

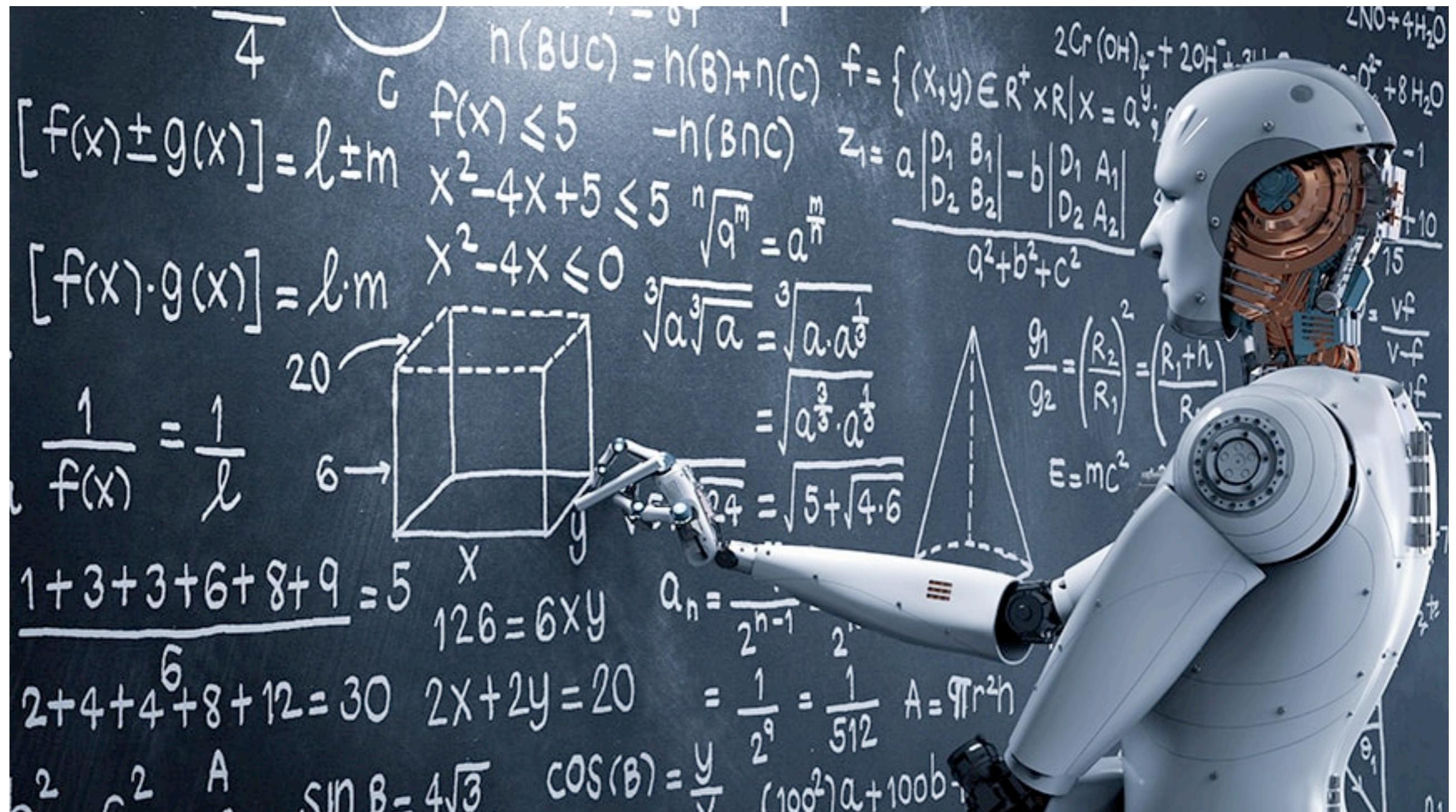
- **Day 5:**

- Unsupervised learning
- Autoencoders
- Generative Adversarial Networks
- Normalizing Flows

*Hands on sessions each day will closely follow the lectures topics*

# Table of contents

- What is machine learning
- Notation
- Supervised Learning
- Linear regression
- Linear classification
- Gradient Descent
- Overfitting
- Performance metrics
- Summary



# What is machine learning?

## The first definition:

Giving computers the ability to learn without explicitly programming them  
[Arthur Samuel, 1959]

# What is machine learning?

### The first definition:

Giving computers the ability to learn without explicitly programming them  
[Arthur Samuel, 1959]

### What it actually involves:

Statistics + Algorithms  
Computer Science + Probability +  
Optimization Techniques

# What is machine learning?

### **The first definition:**

Giving computers the ability to learn without explicitly programming them  
[Arthur Samuel, 1959]

### **What it actually involves:**

Statistics + Algorithms  
Computer Science + Probability +  
Optimization Techniques

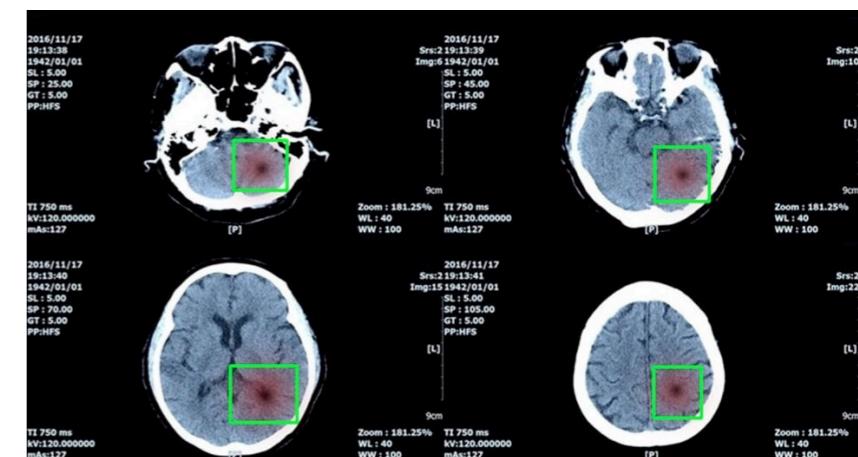
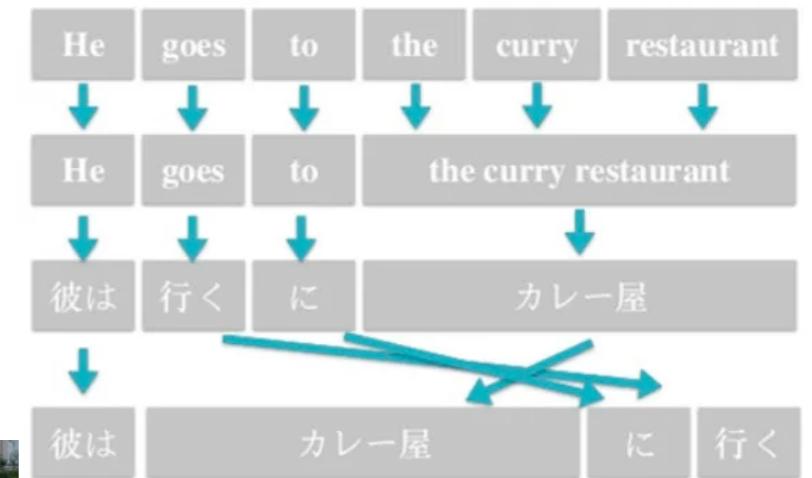
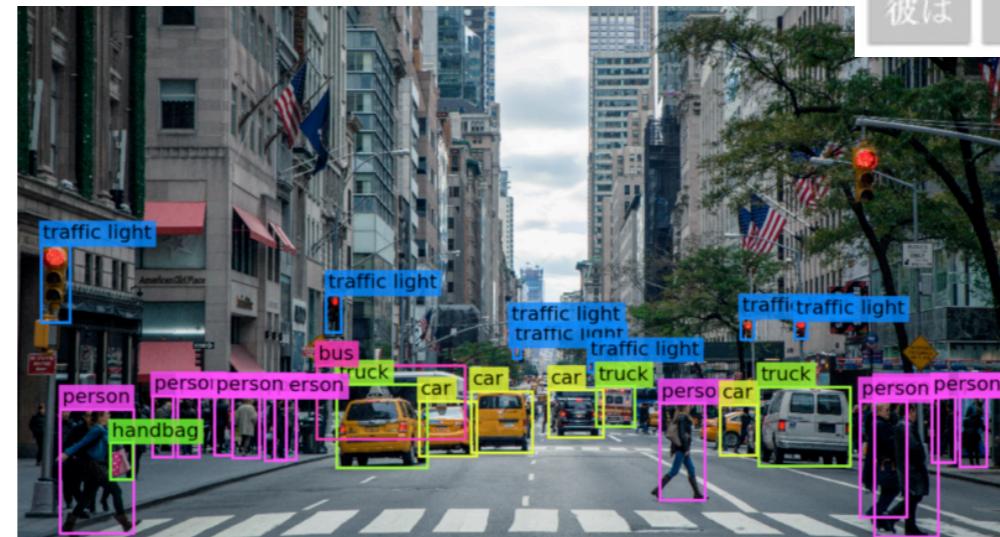
### **How you can more simply think about it:**

Fitting data with complex functions  
Learn from data mathematical models that characterize the patterns,  
regularities, and relationships amongst variables in the system

# **What is machine learning?**

# Where is ML used?

- Natural Language Processing:  
speech and handwriting recognition
- Object recognition and computer vision
- Fraud detection
- Financial market analysis
- Search engines
- Spam and virus detection
- Medical diagnosis
- Robotics control
- Automation: energy usage,  
systems control, video games, self-driving cars
- Advertising
- Data science
- ....



# ML for science

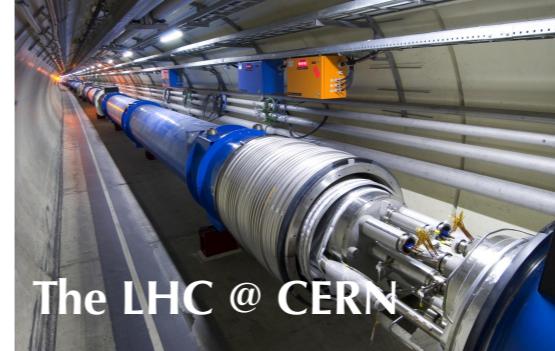
- Classify physics processes/objects
  - Compress granular sensor data to lower dimension interpretable features
  - Simulation
  - Experiment control
  - ...



# Astrophysics & cosmology



# Quantum Information Science



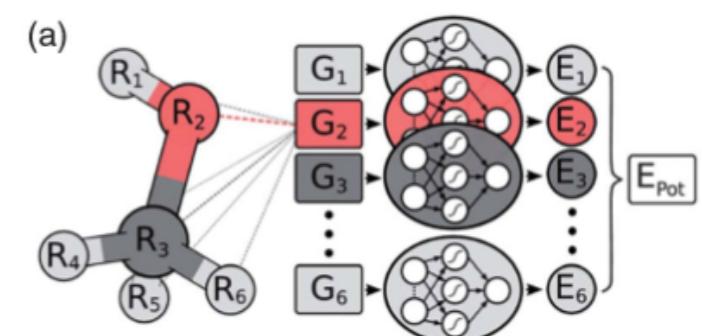
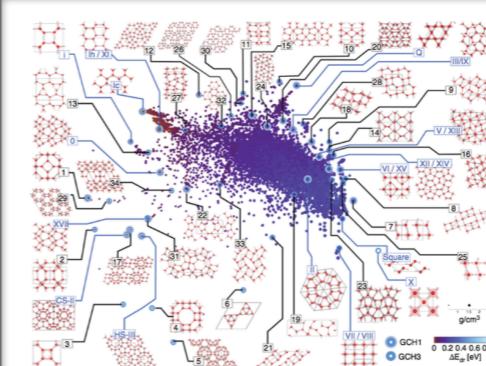
The LHC @ CERN



# The DUNE neutrino experiment

# Particle Physics

$$\begin{aligned}
& \mathcal{R} = \frac{h}{\sqrt{2eUm_e R_0^2 S}} \quad I_d = I_0(1 + d\alpha t) \quad I = \frac{I_0}{R_0} \quad \omega = 2\pi f \\
& \frac{f}{f_0} = \frac{1}{2\pi f_0} \frac{V_{AC}}{\sqrt{L C}} \sin \frac{2\pi f t}{L} \quad I = \frac{I_0}{2} \sqrt{1 - \beta^2} \quad \beta = \frac{\Delta f}{f_0} \quad I_0 = \frac{\Delta E}{m_e V_0} \quad \frac{dE}{dx} = \frac{m_e V_0}{\gamma e B_0} \\
& \vec{B} = \vec{B}_0 + \mu_0 \frac{d\vec{A}}{dt} \quad \vec{S} = \frac{1}{\mu_0} (\vec{E} \times \vec{B}) \quad \Delta I \propto \frac{1}{x} \quad \oint \vec{B} d\vec{l} = Q^* \\
& \frac{1}{m_e} \left( \frac{D_{AT}}{m_e} + \frac{D_{TNA}}{M_e} + \frac{D_{TA}}{M_e M_0} \right) = \frac{1}{M_e} \left( \frac{D_{AT}}{m_e} + \frac{D_{TNA}}{M_e} \right) \\
& \mathcal{Z} = \frac{E_{kin}}{T} F_h = S h \rho g f \quad E = \hbar k^2 \frac{1}{r} \text{pc} = \frac{1AU}{r} \quad S = \frac{U}{F_V} = \frac{1}{F_V} R \\
& \left( \frac{E_k}{E_0} \right)^2 = \frac{2 m_e v_0^2 \cos^2 \theta_0}{c^2} \quad \frac{1}{\sqrt{1 - \frac{v^2}{c^2}}} = \frac{Q}{M} = \frac{F_0}{F_0 + R} \quad \lambda^{**} T = b \\
& \frac{2 \pi}{\lambda} = \frac{2 \pi c \lambda}{c^2} \cos \theta_0 \quad R = R_0 \frac{3 \pi}{4} \frac{1}{\cos \theta_0} \\
& S = \frac{1}{4} \frac{d\sigma}{dt} \quad \int \vec{B} \cdot d\vec{l} = \int \left( \int \frac{\partial \vec{A}}{\partial x} \cdot d\vec{x} \right) d\vec{l} = \int \frac{\partial \vec{A}}{\partial x} \cdot \left( \int d\vec{x} \right) d\vec{l} = \int \frac{\partial \vec{A}}{\partial x} \cdot \vec{l} d\vec{l} \\
& \text{where } F_0 = \frac{q}{2} \frac{B_0^2}{8\pi} \text{ and } \frac{d\sigma}{dt} = \frac{C_0}{\lambda^2} \\
& \oint \vec{B} d\vec{l} = \mu_0 \sum I_i \quad p_i = \frac{E_i}{4\pi} \frac{d\Omega}{d\theta} \quad P = UI \quad k = \frac{2\pi}{\lambda} \quad \nu = \frac{c}{\lambda} (1 + \beta \cos \theta) \quad \frac{dE}{dx} = \frac{m_e V_0}{\gamma e B_0} \\
& C = \frac{\sin(\theta_0 - \theta_0')}{(\sin \theta_0 \sin \theta_0')^2} = \frac{\beta_{\theta_0} - \beta_{\theta_0'}}{(\beta_{\theta_0} \beta_{\theta_0'})^2} \quad \nabla \times (\vec{B} \vec{B}) = \frac{3}{2} (\nabla \vec{B})^2 = \mu_0 \frac{1}{4\pi} \frac{3}{2} (\frac{\partial \vec{B}}{\partial x})^2 = \nabla \times \frac{\partial \vec{B}}{\partial x}
\end{aligned}$$

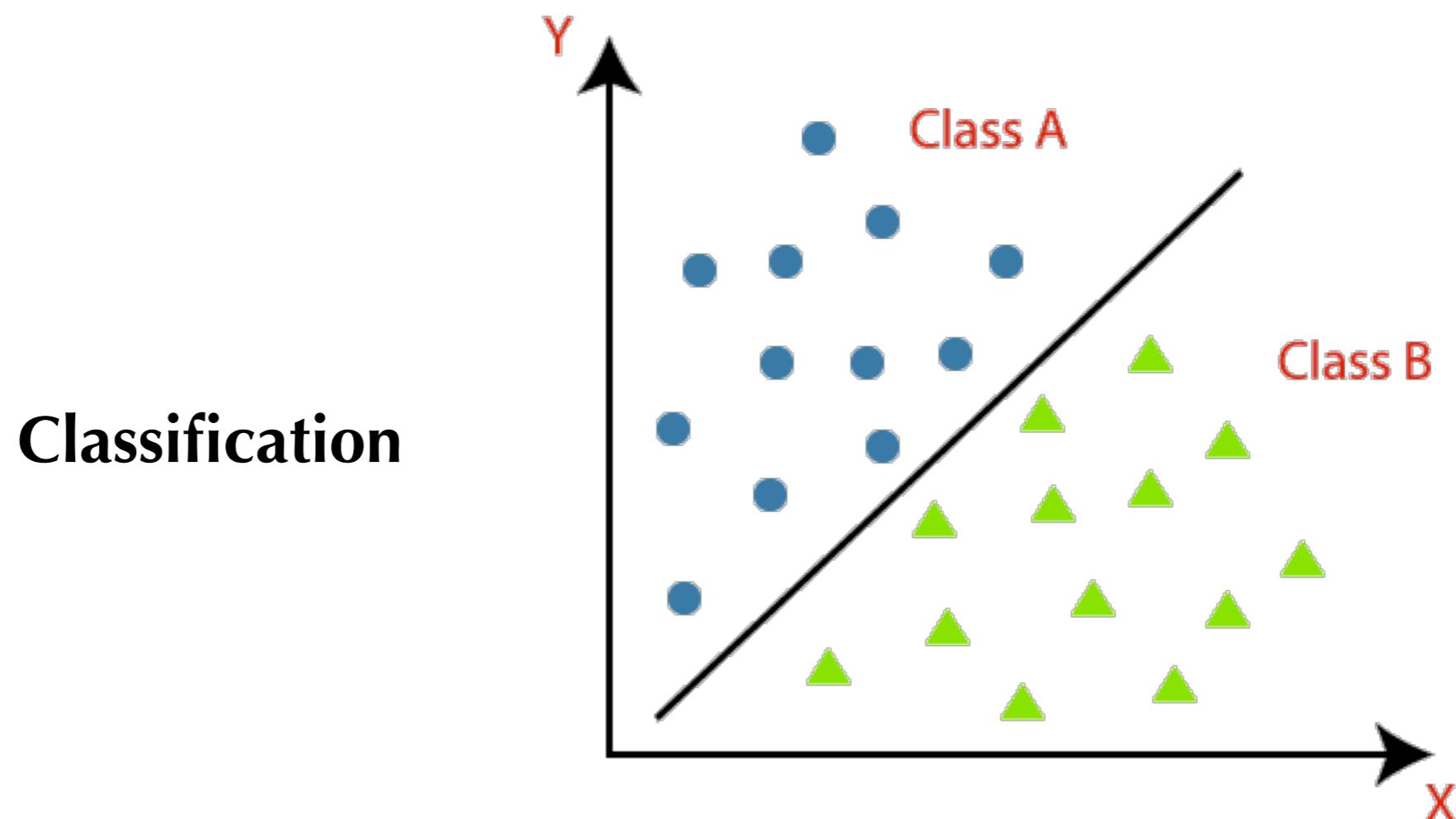


## Chemistry & Materials

Review from G. Carleo et al: [Rev. Mod. Phys. 91, 045002](#)

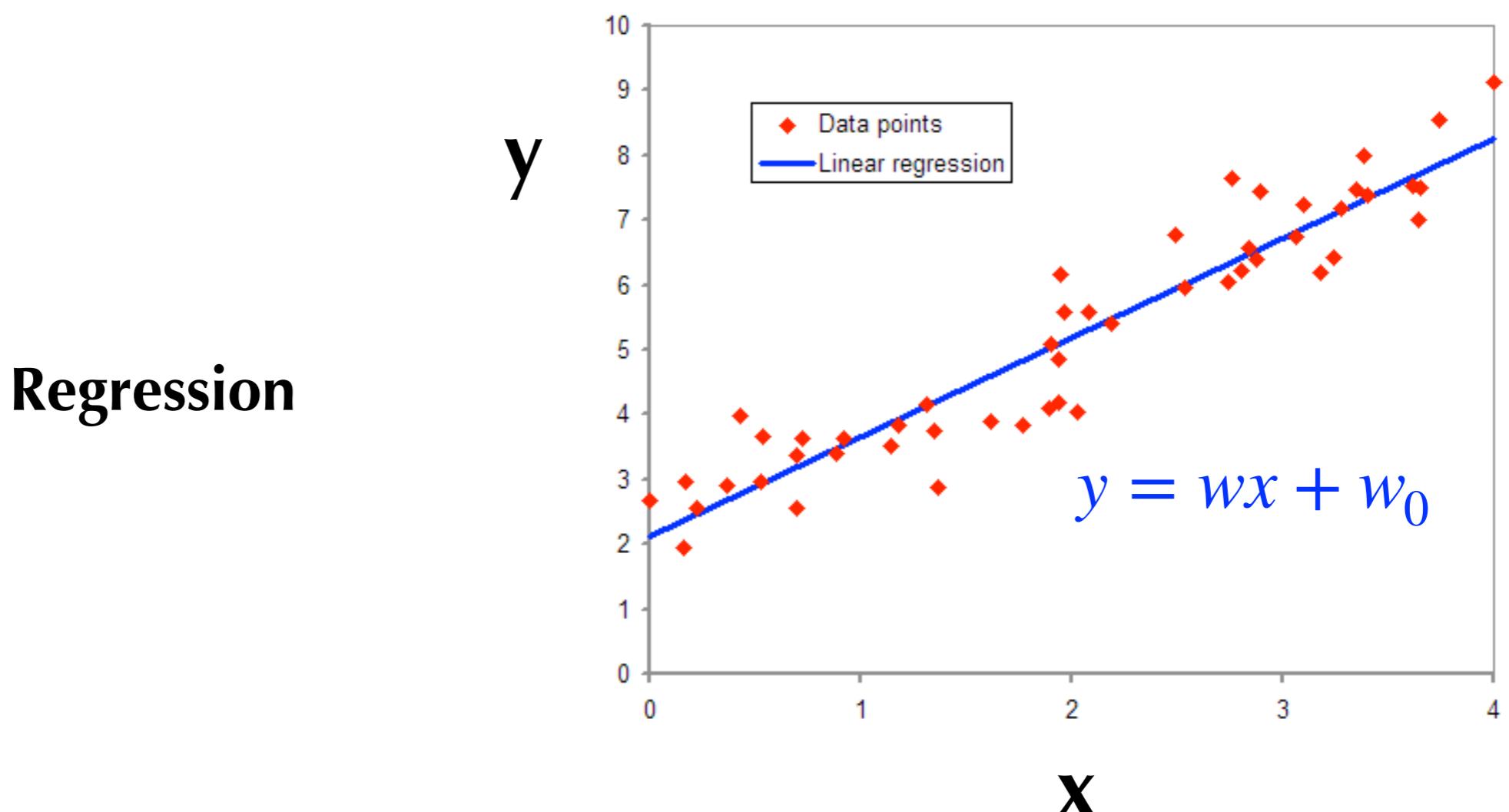
# Machine learning: models

- Key element in machine learning is a **mathematical model**
  - A mathematical characterization of system(s) of interest
  - Chosen model depends on the task and available data



# Machine learning: models

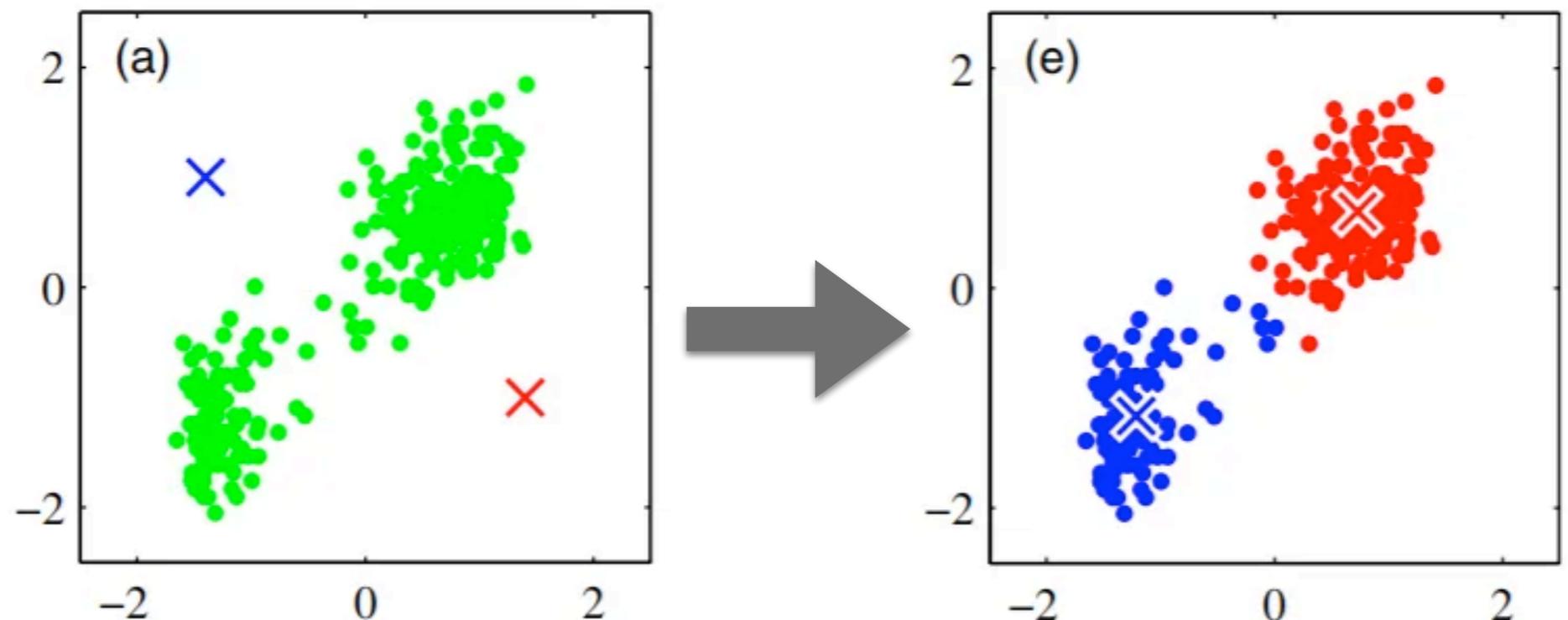
- Key element in machine learning is a **mathematical model**
  - A mathematical characterization of system(s) of interest
  - Chosen model depends on the task and available data



# Machine learning: models

- Key element in machine learning is a **mathematical model**
  - A mathematical characterization of system(s) of interest
  - Chosen model depends on the task and available data

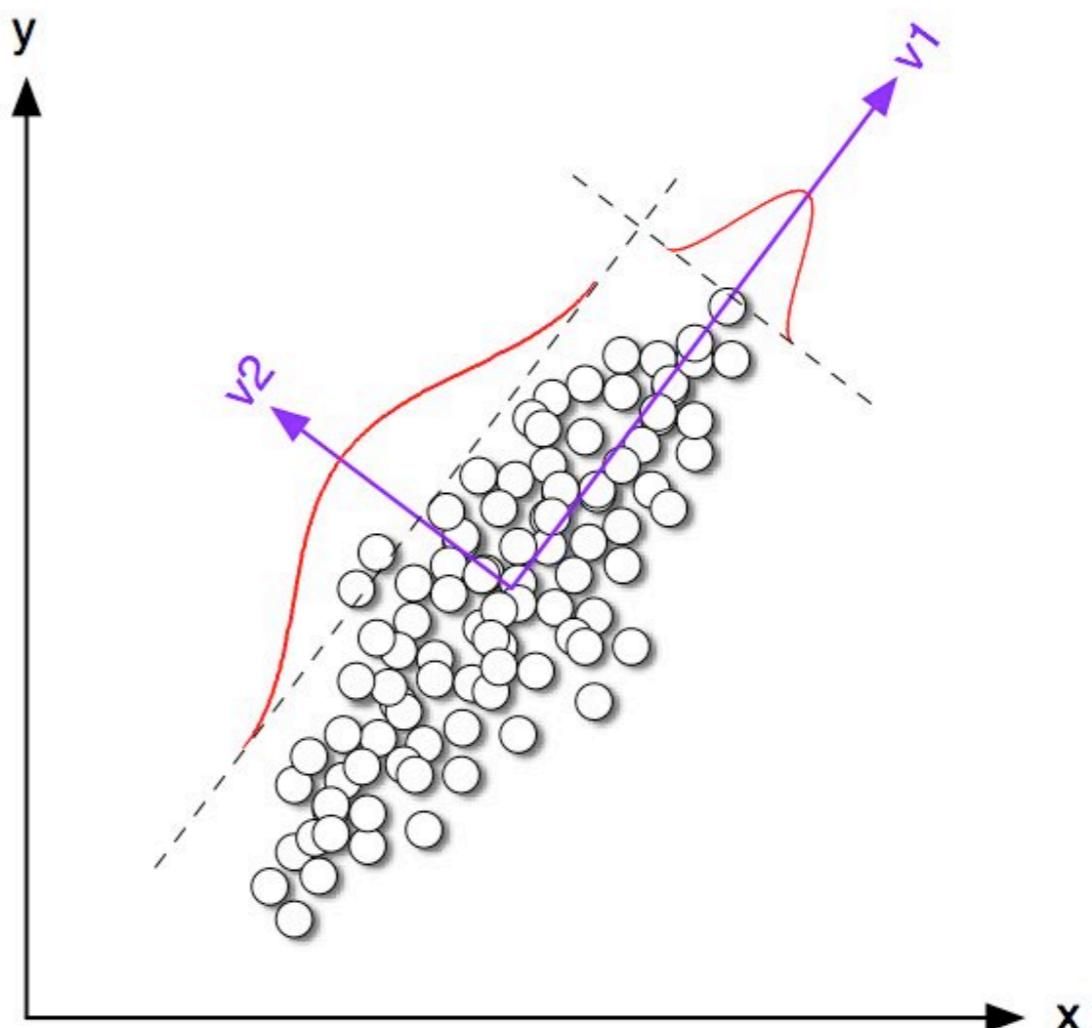
**Clustering**



# Machine learning: models

- Key element in machine learning is a **mathematical model**
  - A mathematical characterization of system(s) of interest
  - Chosen model depends on the task and available data

**Dimensionality Reduction**



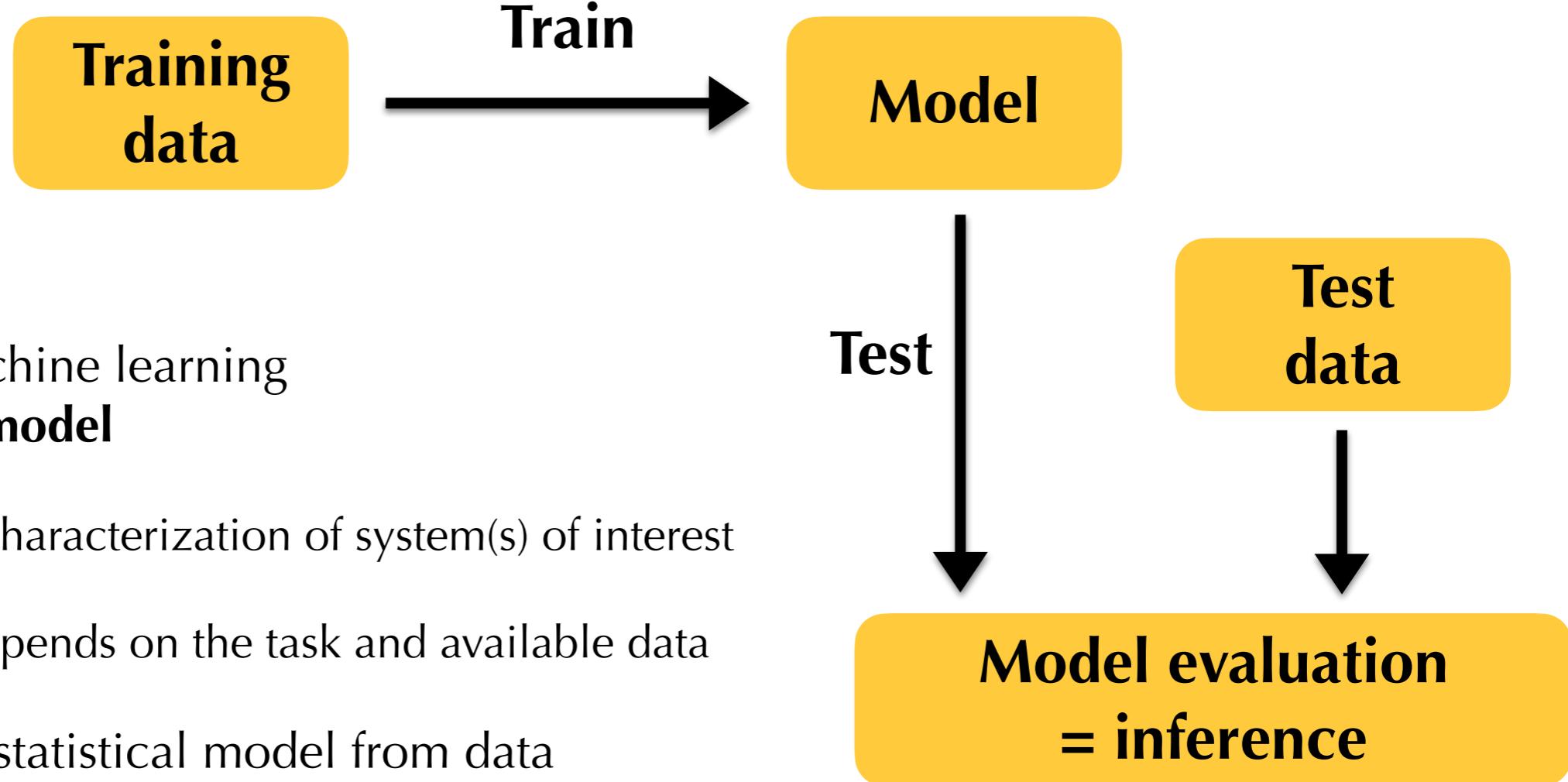
# Machine learning: models



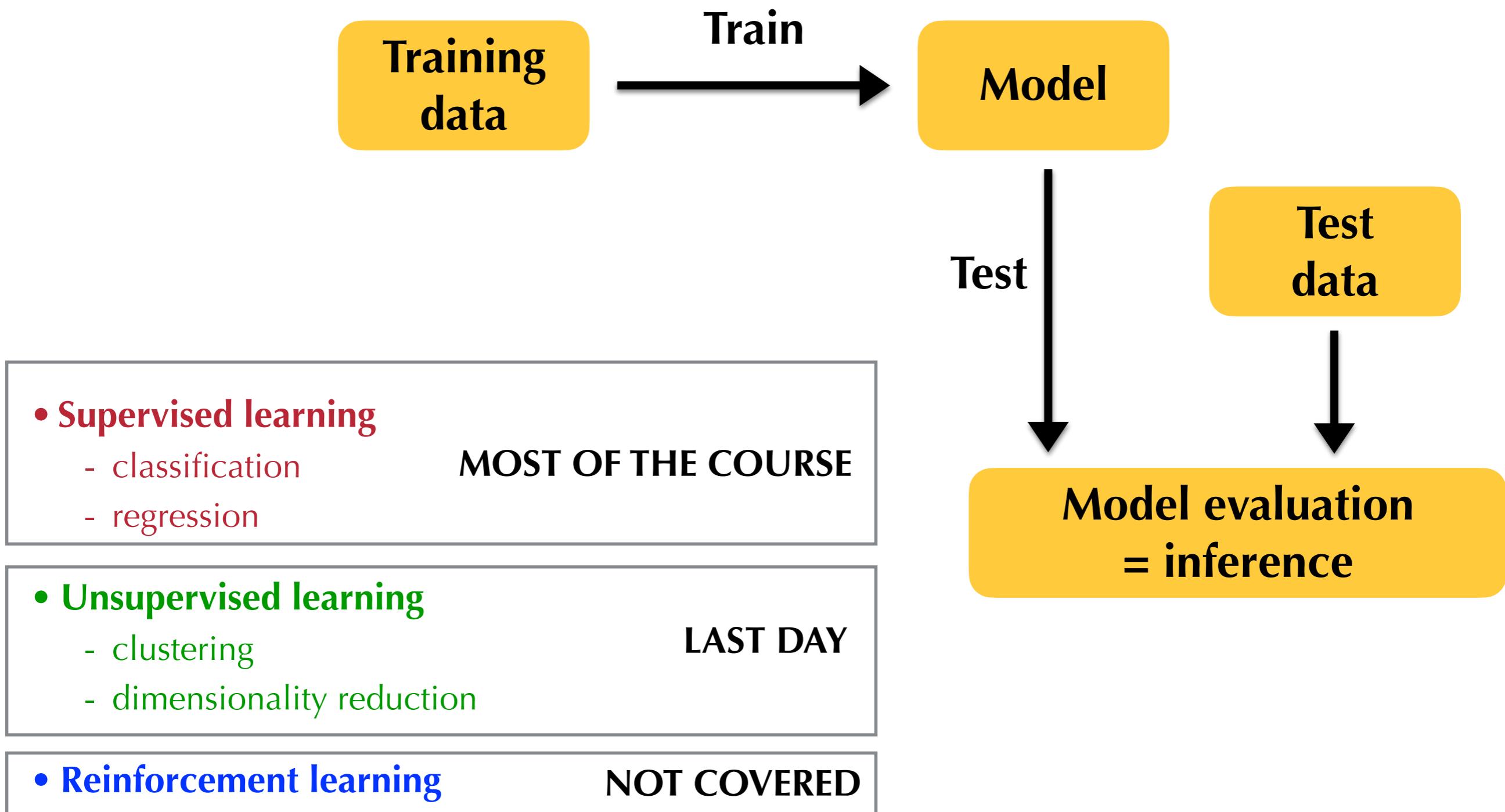
- Key element in machine learning is a **mathematical model**
  - A mathematical characterization of system(s) of interest
  - Chosen model depends on the task and available data
- **Learning:** estimate statistical model from data

# Machine learning: models

- Key element in machine learning is a **mathematical model**
  - A mathematical characterization of system(s) of interest
  - Chosen model depends on the task and available data
- **Learning:** estimate statistical model from data
- **Prediction and Inference:** using statistical model to make predictions on new data points and infer properties of system(s)



# Types of learning



# Notation

---

# Notation

- $\mathbf{X} \in \mathbb{R}^{m \times n}$  Matrices in bold upper case
- $\mathbf{x} \in \mathbb{R}^n$  Vectors in bold lower case
- $x \in \mathbb{R}$  Scalars in lower case, non bold
- $\mathcal{X}$  Sets are upper case cursive
- $\{\mathbf{x}_i\}_{i=1,\dots,m}$  Sequence of vectors  $\mathbf{x}_1, \dots, \mathbf{x}_m$
- $y \in \mathbb{I}^k / \mathbb{R}^k$  Labels represented as
  - a) integers for classes or
  - b) real numbers for regression
- Variables = features = inputs
- Data point  $\mathbf{x} = \{x_1, \dots, x_n\}$  has  $n$  features
- Typically use affine coordinates:  
$$y = \mathbf{w}^T \mathbf{x} + w_0 \longrightarrow \mathbf{w}^T \mathbf{x}$$
$$\longrightarrow \mathbf{w} = \{w_0, \dots, w_n\}$$
$$\longrightarrow \mathbf{x} = \{x_0, \dots, x_n\}$$

# Probability review

•  $p(x, y)$  Joint distribution of two variables

•  $p(x) = \int p(x, y) dy$  Marginal distribution

•  $p(y|x) = \frac{p(x,y)}{p(x)}$  Conditional probability

•  $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$  Bayes theorem

•  $E[f(x)] = \int f(x)p(x)dx$  Expected value

•  $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2} \frac{(x-\mu)^2}{\sigma^2}\right)$  Normal distribution  $N(\mu, \sigma)$

# **Supervised Learning**

---

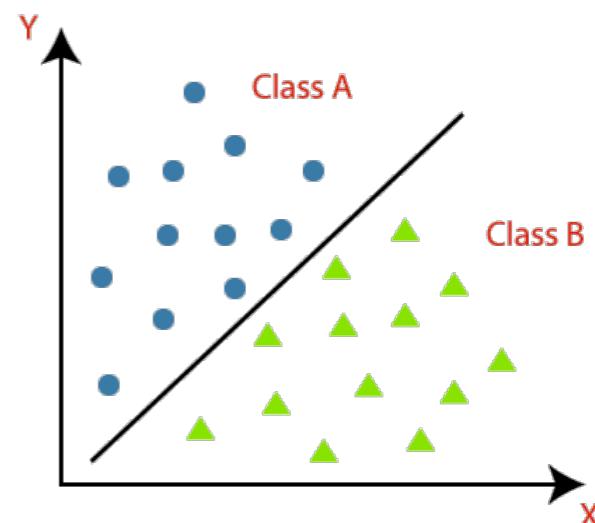
# Supervised learning

- Given  $N$  examples with features  $\{x_i \in \mathcal{X}\}$  and targets  $\{y_i \in \mathcal{Y}\}$ , learn function mapping  $h(x) = y$

- **Classification:**  $\mathcal{Y}$  is a finite set of labels (i.e. classes)

$\mathcal{Y} = \{0,1\}$  for binary classification,  
encoding classes, e.g. Higgs boson (1) vs background (0)

$\mathcal{Y} = \{c_1, c_2, \dots, c_n\}$  for multi-class classification  
represent with “**one-hot-vector**”:  $y_i = (0,0,\dots,1,\dots,0)$   
where  $k^{\text{th}}$  element is 1 and all others zero for class  $c_k$



# Supervised learning

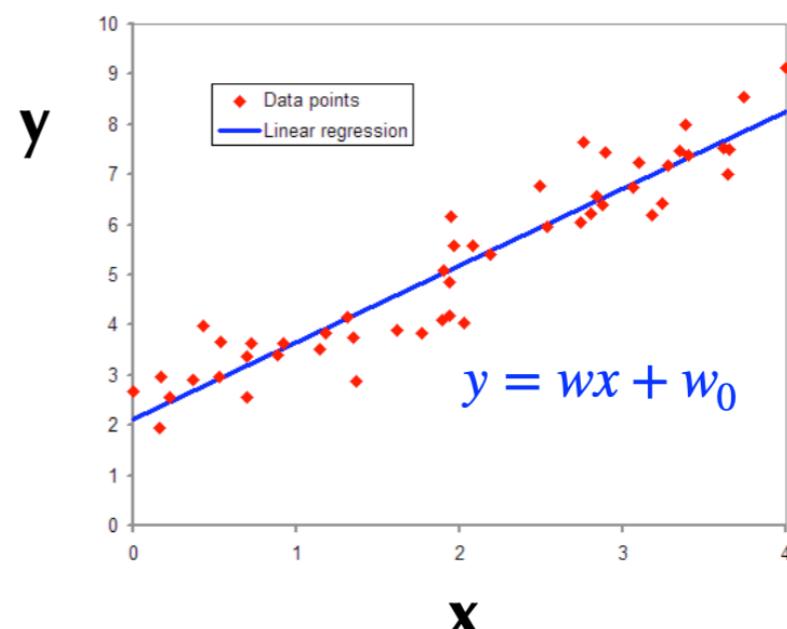
- Given  $N$  examples with features  $\{x_i \in \mathcal{X}\}$  and targets  $\{y_i \in \mathcal{Y}\}$ , learn function mapping  $h(x) = y$

- **Classification:**  $\mathcal{Y}$  is a finite set of labels (i.e. classes)

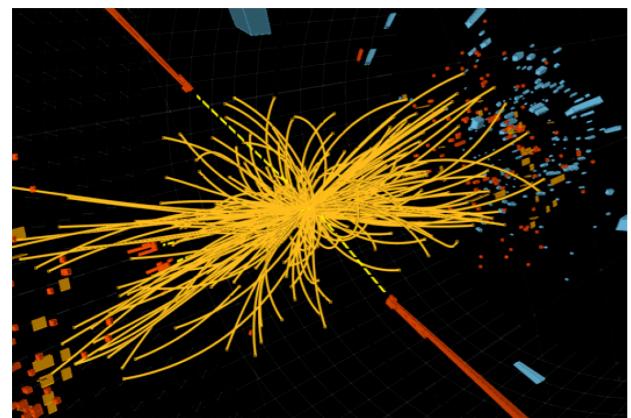
$\mathcal{Y} = \{0,1\}$  for binary classification,  
encoding classes, e.g. Higgs boson (1) vs background (0)

$\mathcal{Y} = \{c_1, c_2, \dots, c_n\}$  for multi-class classification  
represent with “**one-hot-vector**”:  $y_i = (0,0,\dots,1,\dots,0)$   
where  $k^{\text{th}}$  element is 1 and all others zero for class  $c_k$

- **Regression:**  $\mathcal{Y}$  is a real number  
(eg., the truth mass of the SM Higgs boson)



# Supervised Learning: how does it work?

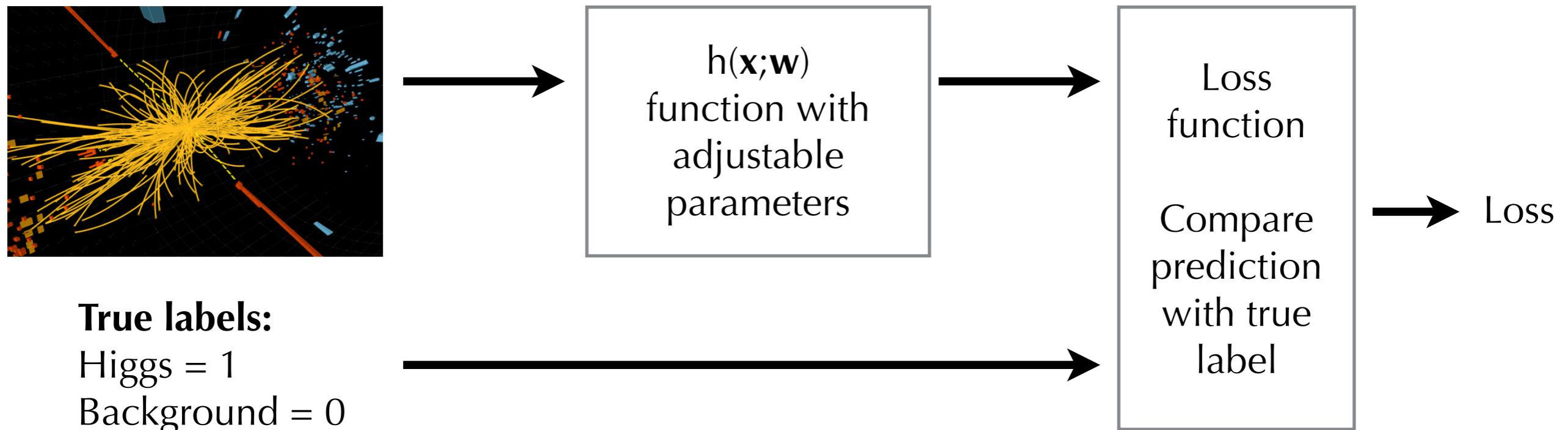


**True labels:**

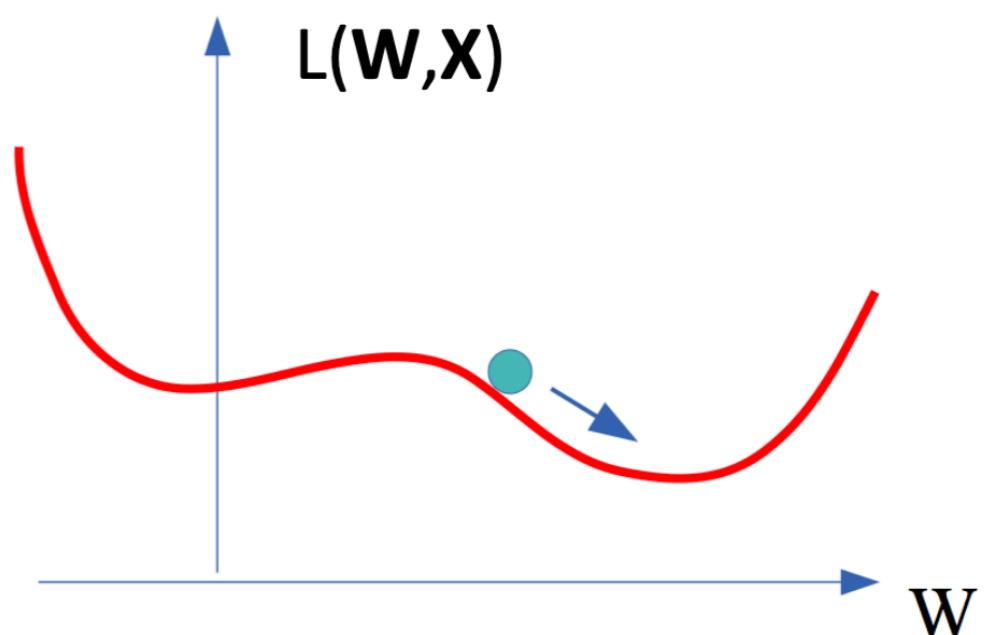
Higgs = 1

Background = 0

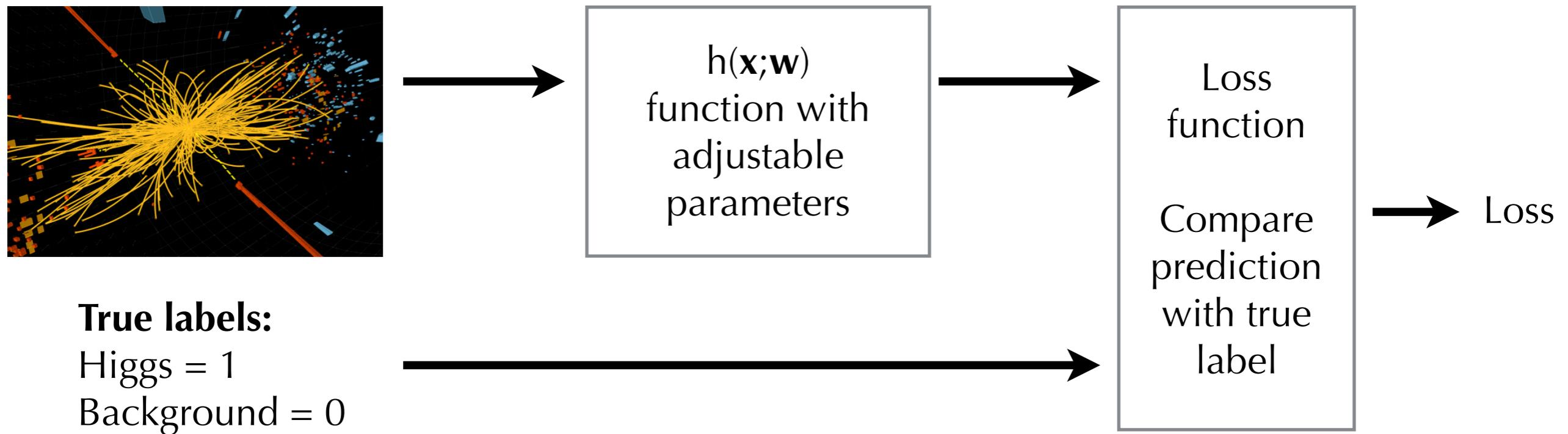
# Supervised Learning: how does it work?



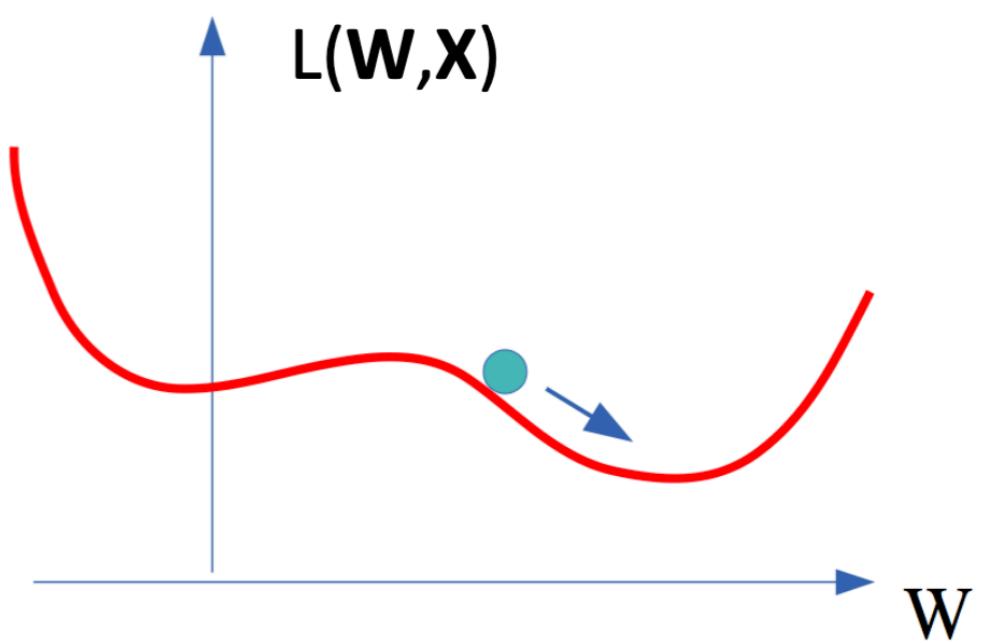
- Design function with adjustable parameters
- Design a Loss function



# Supervised Learning: how does it work?



- Design function with adjustable parameters
- Design a Loss function
- Find best parameters which minimize the loss
  - Use a labeled training-set to compute loss
  - Adjust parameters to reduce loss function
  - Repeat until parameters stabilize
- Estimate final performance on test dataset (independent from training set)



# Empirical risk minimization

$$\arg \min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^N L(h(\mathbf{x}_i; \mathbf{w}), y_i) + \lambda \Omega(\mathbf{w})$$

Average expected loss

Model regularization

- Framework to design learning algorithms
  - $L(\dots)$  is a **loss function** comparing prediction  $h(\dots)$  with target  $y$
  - $\Omega(\mathbf{w})$  is a **regularizer**, penalizing certain values of  $\mathbf{w}$ 
    - $\lambda$  controls how much we penalize, and is a **hyperparameter** that we have to tune
    - more on this later
- Learning is cast as an optimization problem

# Example loss functions

- **Square error loss**

- often used in regression

$$L(h(\mathbf{x}; \mathbf{w}), y) = (h(\mathbf{x}; \mathbf{w}) - y)^2$$

- **Cross entropy**

- with  $y \in \{0,1\}$

$$L(h(\mathbf{x}; \mathbf{w}), y) = -y \log h(\mathbf{x}; \mathbf{w}) - (1 - y) \log(1 - h(\mathbf{x}; \mathbf{w}))$$

- often used in classification

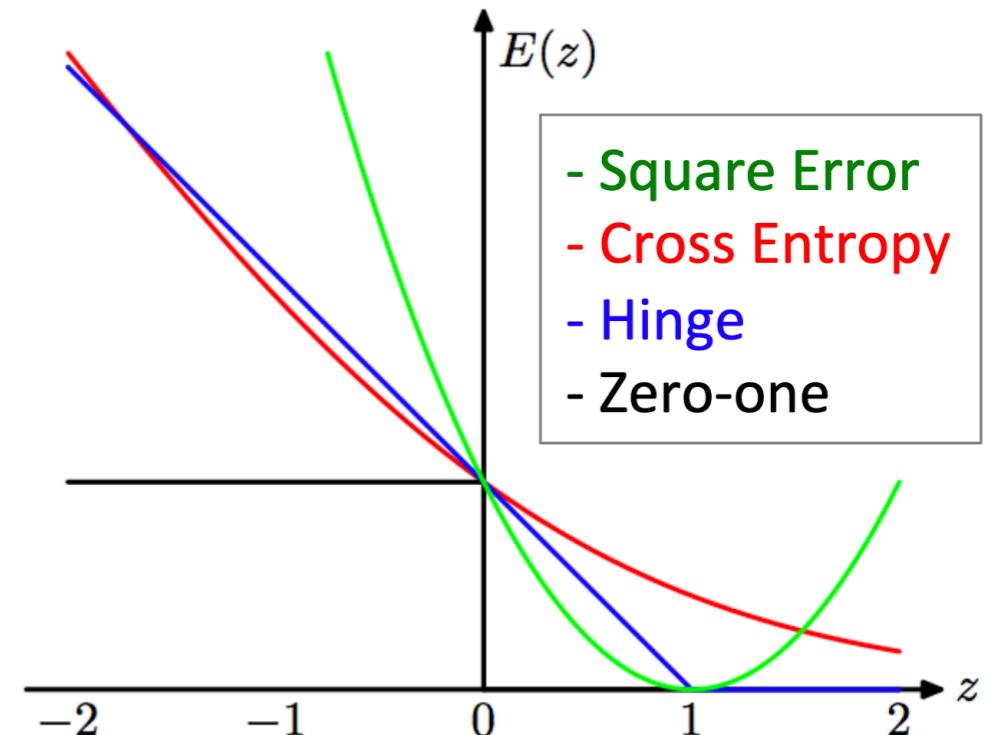
- **Hinge Loss (*not in this course*)**

- with  $y \in \{-1,1\}$

$$L(h(\mathbf{x}; \mathbf{w}), y) = \max(0, 1 - yh(\mathbf{x}; \mathbf{w}))$$

- **Zero-One loss (*not in this course*)**

- with  $h(\mathbf{x}; \mathbf{w})$  predicting label



*Can be derived from probability theory – trust me at the moment but ask me if interested!*

# Linear Regression

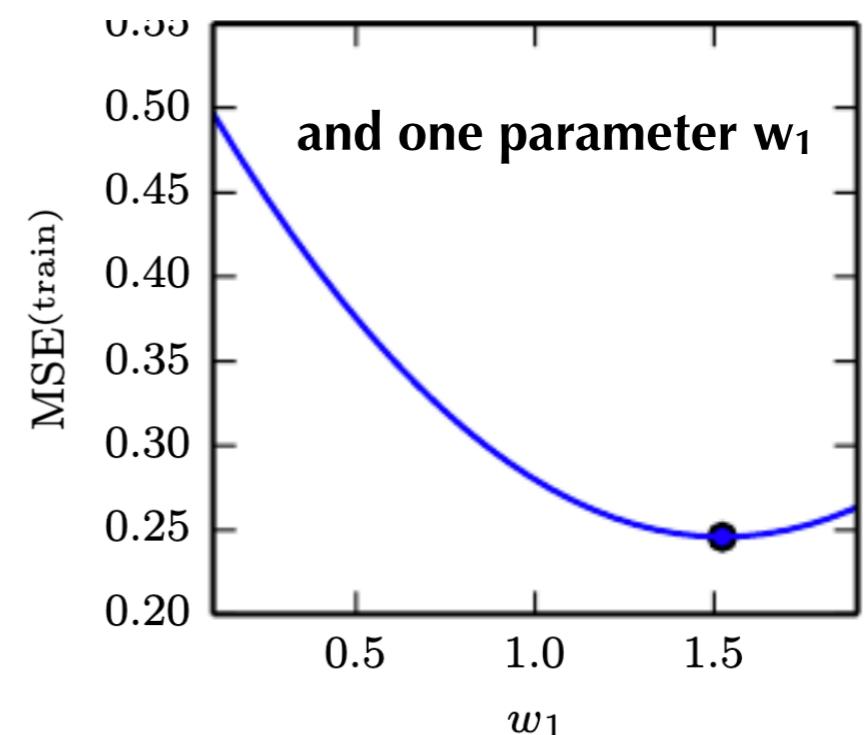
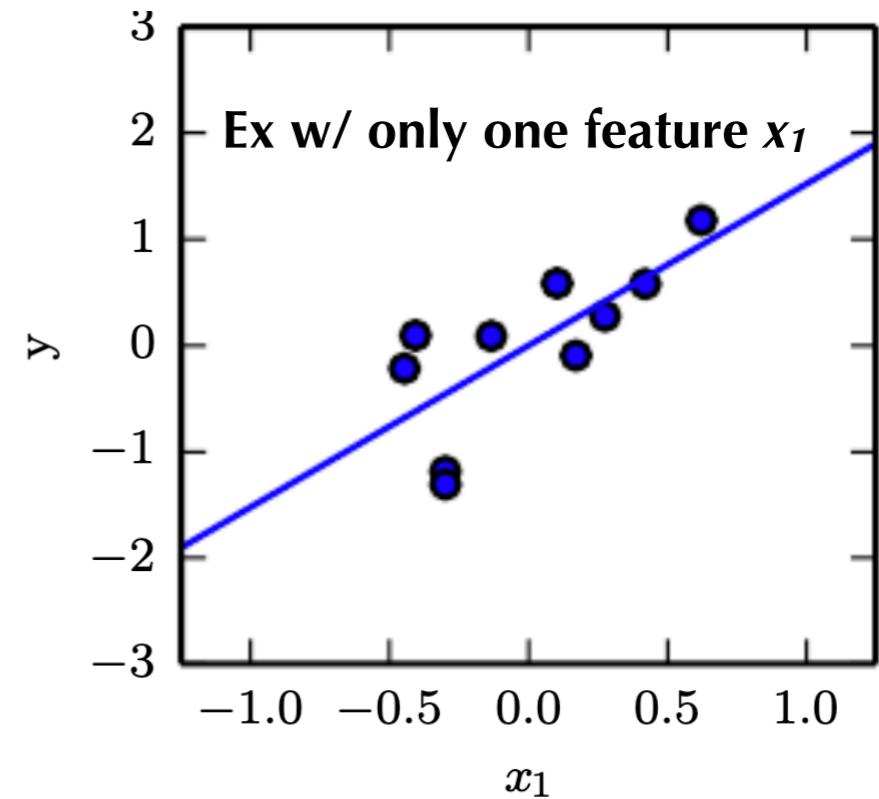
---

# Linear regression

- Set of input / output pairs  $D = \{\mathbf{x}_i, y_i\}_{i=1,\dots,n}$  where  $n$  is number of examples
  - $\mathbf{x}_i \in \mathbb{R}^m$  ( $m$  = number of features)
  - $y_i \in \mathbb{R}$  (just one target per example)
- Assume a **linear model**:  $\hat{y} = h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$
- To find parameters  $w$  of  $h$  that best fit the data we chose the **Mean Squared Error (MSE)\*** as performance metric (or loss function)

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (y_i - h(\mathbf{x}_i; \mathbf{w}))^2$$

- Find  $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \mathcal{L}(\mathbf{w})$



\* equivalent to the least square method – this is not the only method possible and other metrics can be used

# Linear regression: matrix form

- Set of input / output pairs  $D = \{x_i, y_i\}_{i=1,\dots,n}$  where  $n$  is number of examples
  - Design matrix  $\mathbf{X} \in \mathbb{R}^{n \times m}$  ( $m$  = number of features)
  - Target vector:  $\mathbf{y} \in \mathbb{R}^n$  (just one target per example)

• Rewrite linear model as:

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \quad \text{predicted vector} \longrightarrow \hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$$

- Rewrite loss as:  $\mathcal{L}(\mathbf{w}) = ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2 = \frac{1}{n}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$
- Minimization solved by computing gradient:  $\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = 2(\mathbf{y} - \mathbf{X}\mathbf{w})\mathbf{X} = 0$
- The solution can be solved *analytically*\*:  $\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$

\* not always possible or most efficient (we come back to this later)

# Linear Classification

---

# Linear classification

- Task: learn a function to **separate different classes of data** — for two classes we call this “binary classification”:

- $\mathbf{x}_i \in \mathbb{R}^m$
- $y_i \in \{-1, 1\}$  — we call these “true labels”

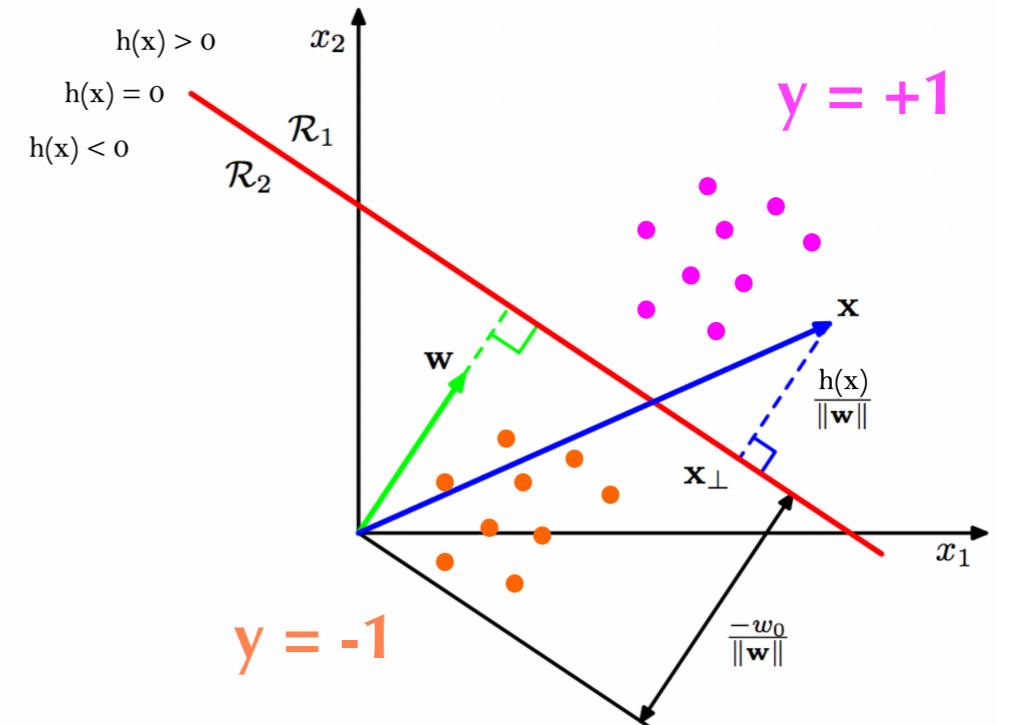
- Assume **linear discriminant function**:  $h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$

- **Decision boundary** defined by hyperplane  
 $h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x} = 0$

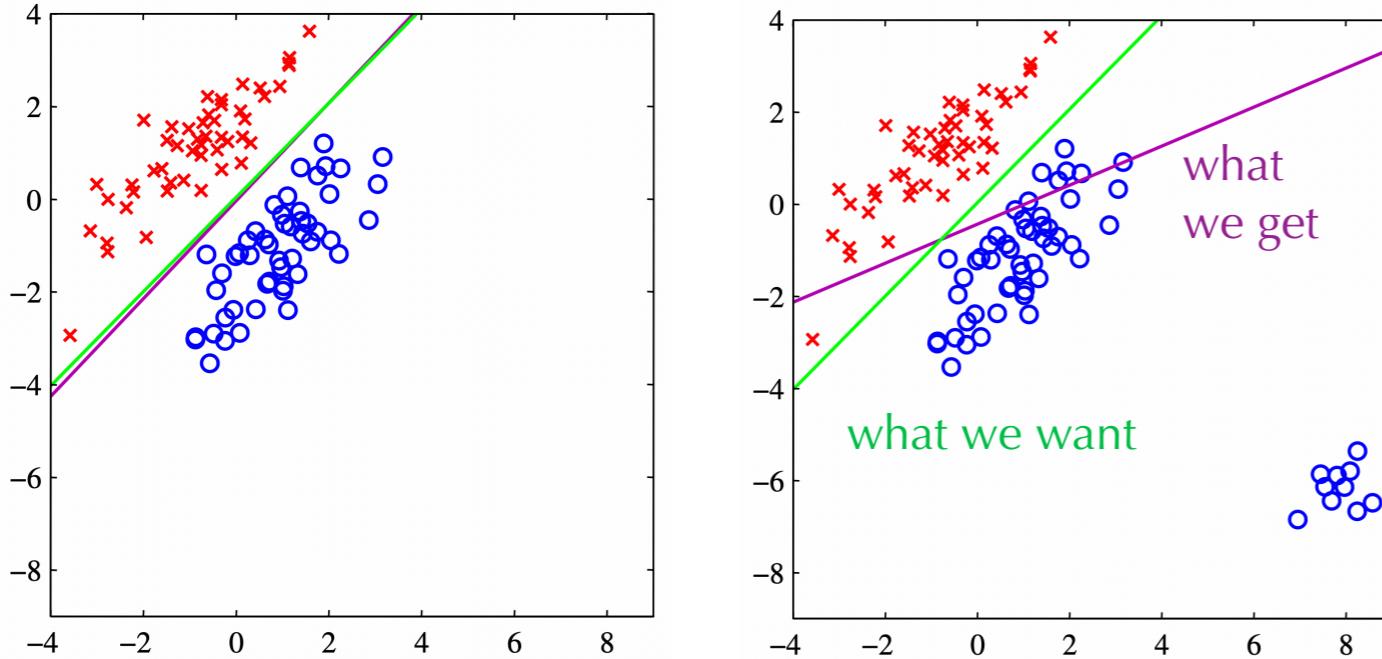
- boundary is perpendicular to weight vector  $\mathbf{w}$

- Classifier score for point  $\mathbf{x}_i = h(\mathbf{x}_i; \mathbf{w}) \rightarrow$  make predictions:

- class = 0 if  $h(\mathbf{x}_i; \mathbf{w}) < 0$
- class = 1 if  $h(\mathbf{x}_i; \mathbf{w}) \geq 0$



# Linear classification

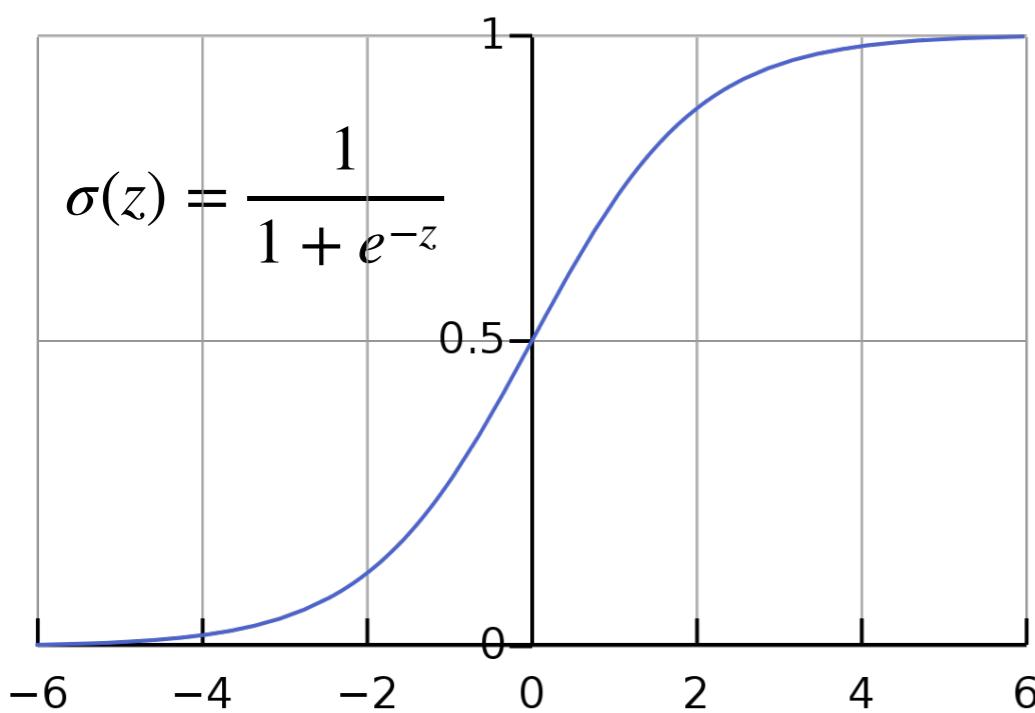


$$\mathcal{L}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (y_i - h(\mathbf{x}_i; \mathbf{w}))^2$$

- As for regression we can apply the least square method to find the decision boundary
  - penalized even when predict class correctly
  - very sensitive to outliers
- The MSE is not the right loss function to use for classification task

# Logistic regression

- Set of input/out pairs  $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1,\dots,n}$ 
  - $\mathbf{x}_i \in \mathbb{R}^m$
  - $y_i \in \{0,1\}$
- Assume linear discriminant function:  $h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$
- Convert the distance from the boundary into a probability that a point belongs to a class or the other → **sigmoid function\***

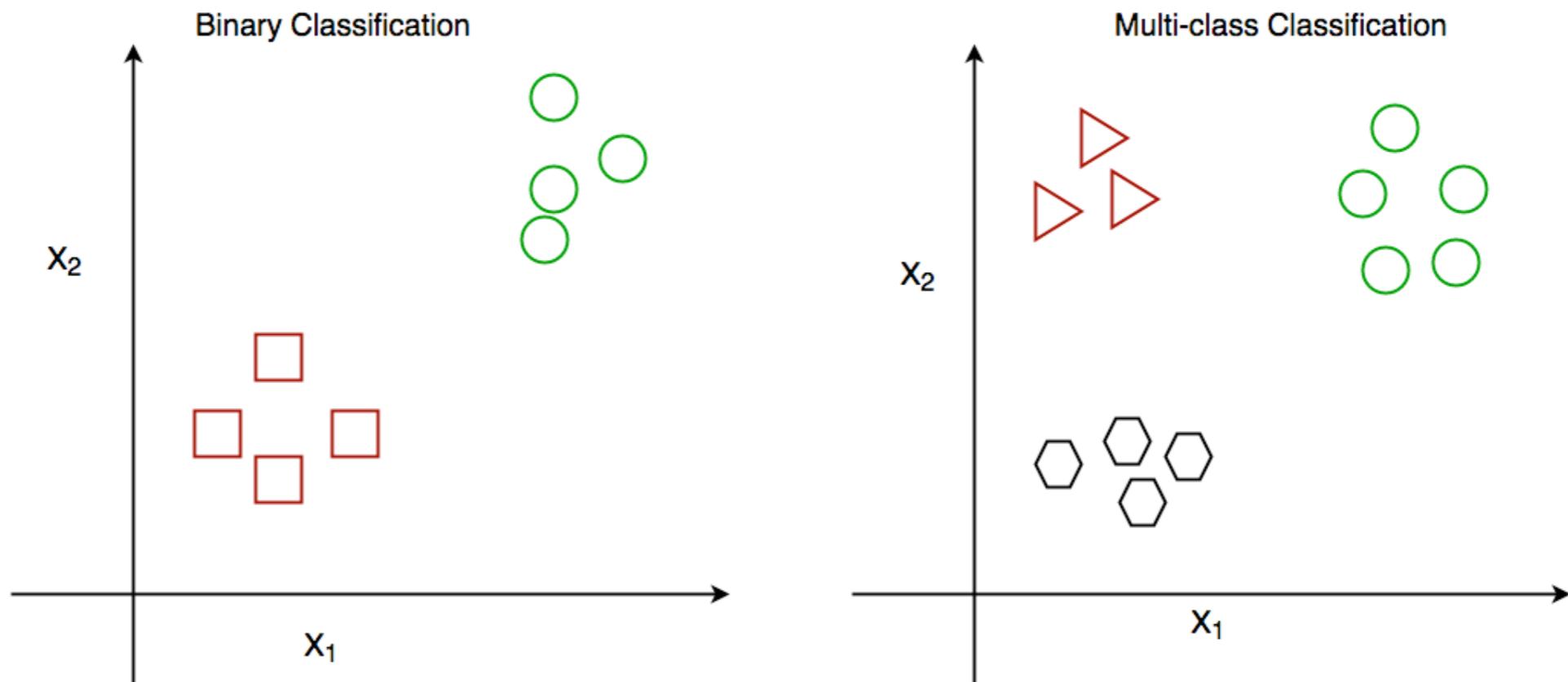


$$p(y = 1 | \mathbf{x}) \equiv p_i = \frac{1}{1 + e^{-h(\mathbf{x}; \mathbf{w})}} = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}$$

- The further from boundary  $\mathbf{w}^T \mathbf{x} = 0$  the more certain about a class
- Class decision rule: choose class 0 if  $p_i < 0.5$ , else choose class 1

\*nb, not a random choice – see generalized linear models

# Multi-class classification



- **Softmax** → multi-class generalization of logistic sigmoid function
  - have  $N$  classes  $\{c_1, \dots, c_N\}$  per each example  $\mathbf{x}_i$  where  $\mathbf{x}_i$  belongs to just one class  $k$
  - model target  $\mathbf{y}_i = \{0, \dots, 0, 1, 0, \dots, 0\} \rightarrow$  set 1 for element  $k_{th}$

$$p(c_k | \mathbf{x}) = \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{j=1}^N e^{\mathbf{w}_j^T \mathbf{x}}}$$

# Logistic regression

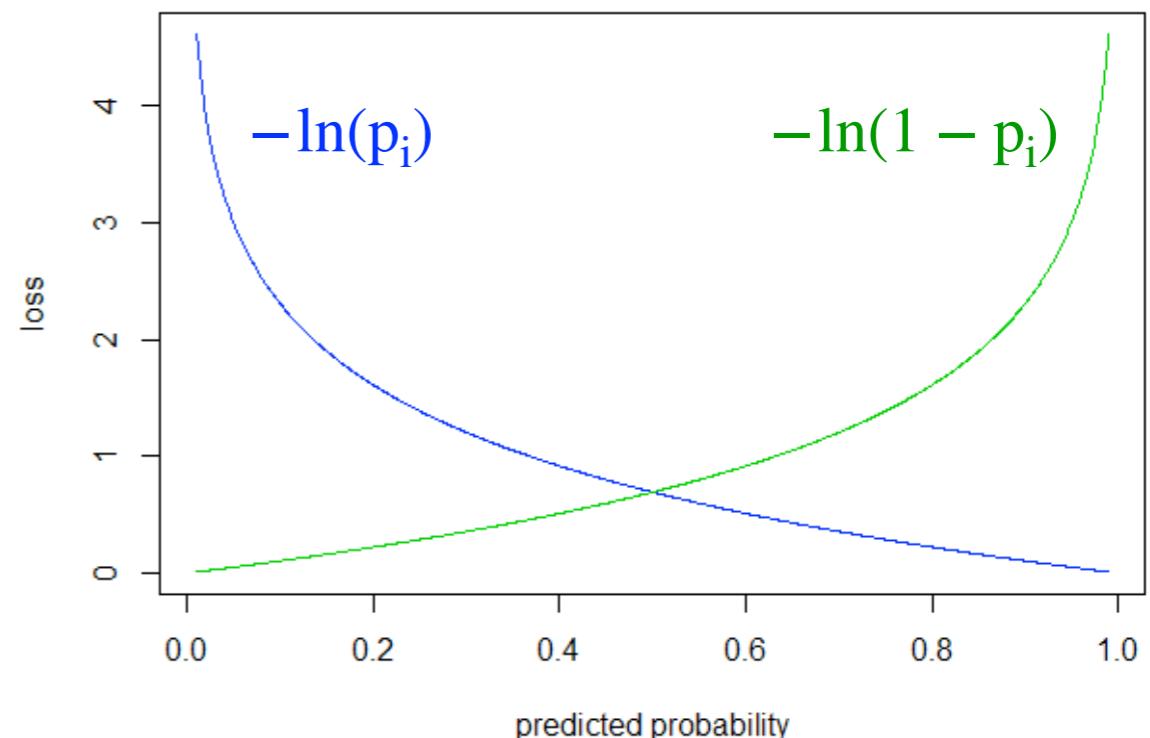
- We now write  $p(y | \mathbf{x})$  as Bernoulli random variable:

$$p(y_i = y | \mathbf{x}_i) = (p_i)^{y_i} (1 - p_i)^{1-y_i} = \begin{cases} p_i & \text{if } y_i = 1 \\ 1 - p_i & \text{if } y_i = 0 \end{cases}$$

- Negative log-likelihood

$$\begin{aligned} -\ln \mathcal{L} &= -\ln \prod_i (p_i)^{y_i} (1 - p_i)^{1-y_i} \\ &= -\sum_i y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i) \end{aligned}$$

**binary cross entropy loss function!**



# Logistic regression

- We now write  $p(y | \mathbf{x})$  as Bernoulli random variable:

$$p(y_i = y | \mathbf{x}_i) = (p_i)^{y_i} (1 - p_i)^{1-y_i} = \begin{cases} p_i & \text{if } y_i = 1 \\ 1 - p_i & \text{if } y_i = 0 \end{cases}$$

- Negative log-likelihood

$$\begin{aligned} -\ln \mathcal{L} &= -\ln \prod_i (p_i)^{y_i} (1 - p_i)^{1-y_i} \\ &= -\sum_i y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i) \quad \text{binary cross entropy loss function!} \\ &= \sum_i \ln(1 + e^{-\mathbf{w}^T \mathbf{x}}) + (1 - y_i) \ln(1 + e^{\mathbf{w}^T \mathbf{x}}) \end{aligned}$$

- No analytic solution to  $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} -\ln \mathcal{L}$
- How to solve for  $\mathbf{w}$  ?
- Many numerical optimization methods to solve — we'll learn today about the most popular **Gradient Descent** method

# Gradient Descent

---

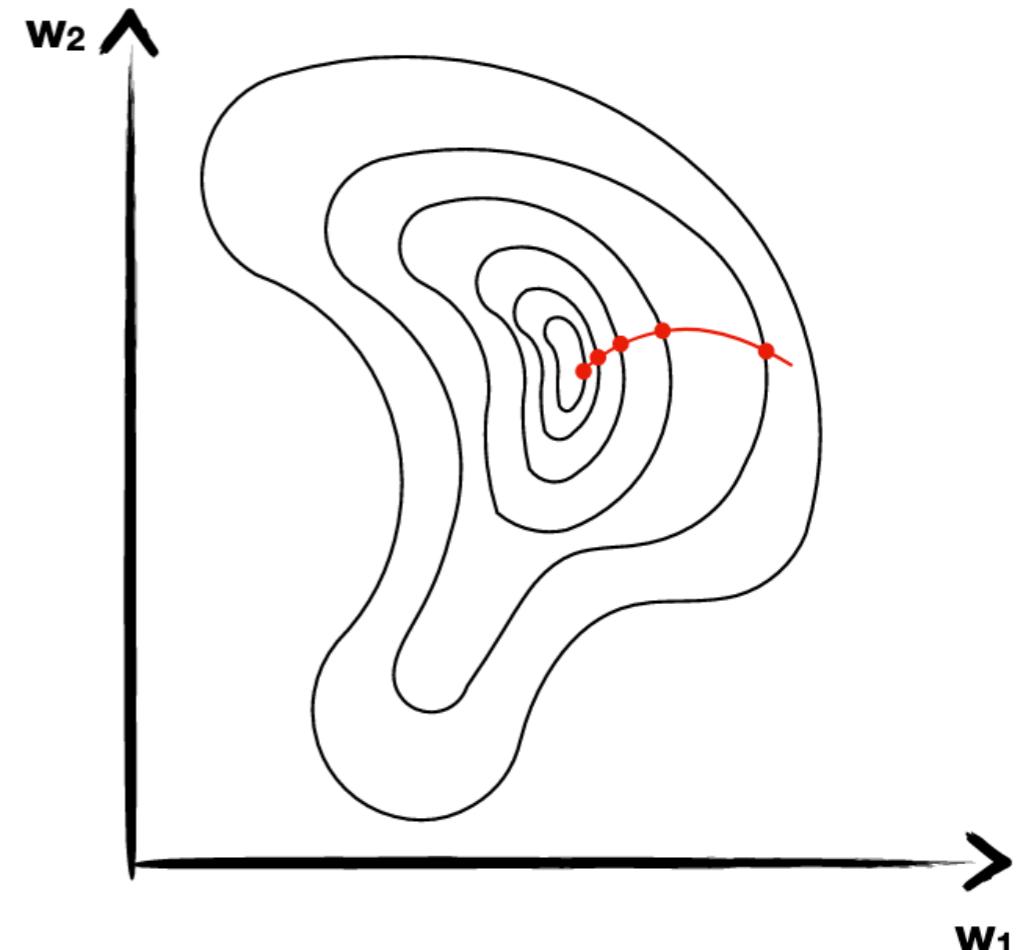
# Gradient Descent

- Start from a random point
- Compute the gradient of the loss wrt the model parameters (i.e. “how should the weight be updated to decrease the loss?”)

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

- Make a step of size  $\eta$  (the **learning rate**) towards the gradient direction
- Update the parameters of the model accordingly

$$\mathbf{w}' \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

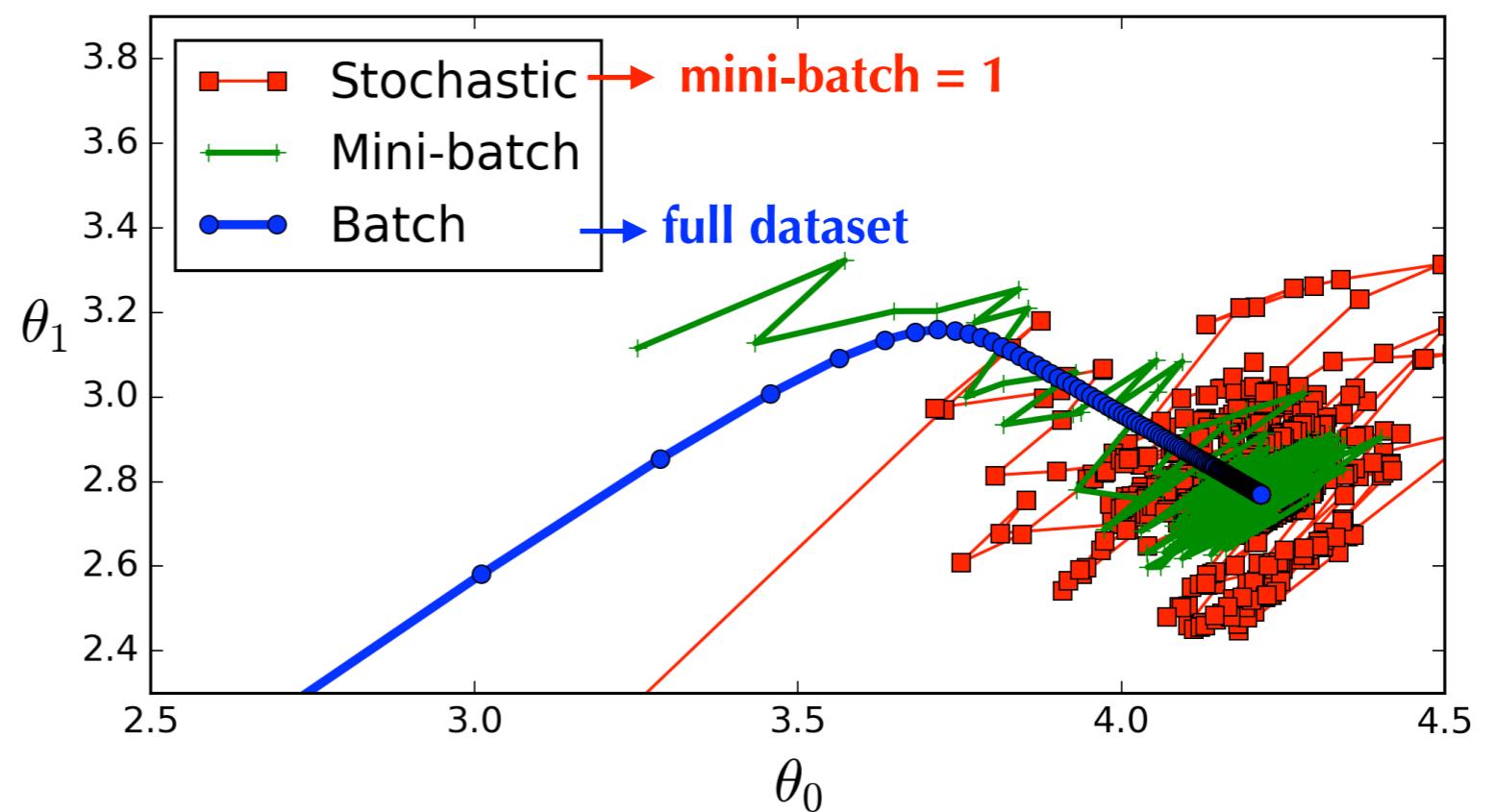


- Effective, but computationally expensive (gradient over entire dataset)

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w})$$

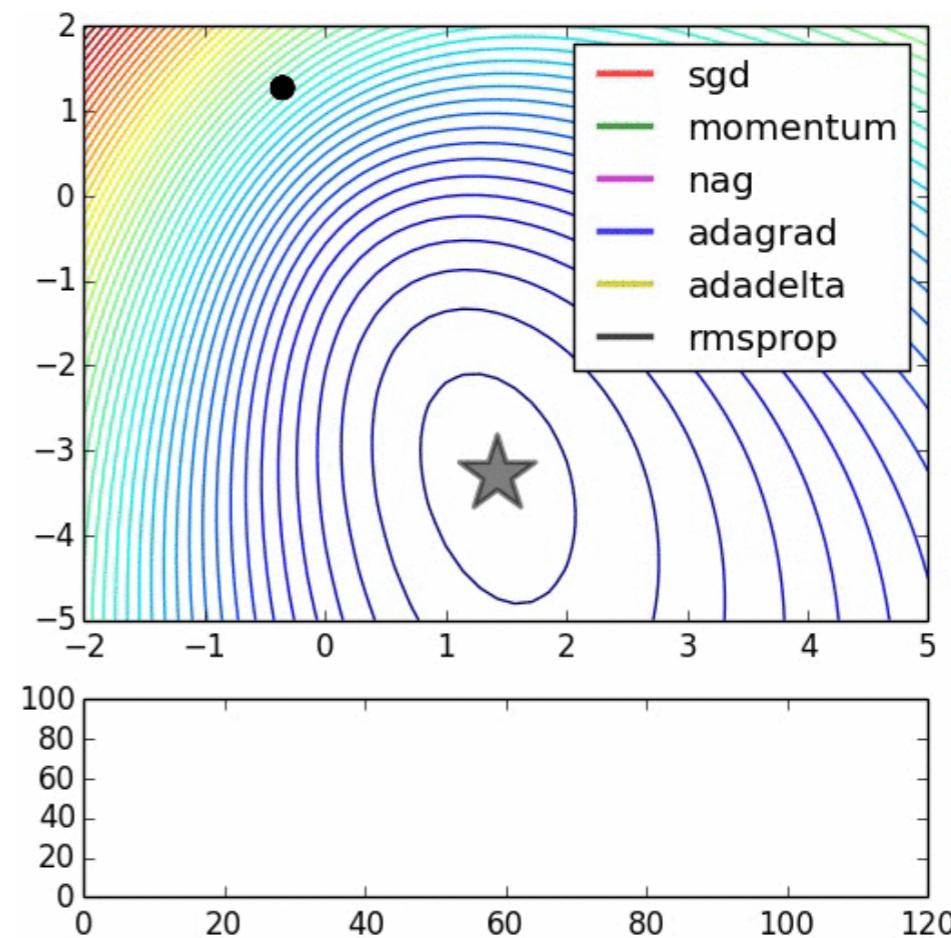
# Stochastic Gradient Descent

- Make the minimization more computationally efficient
  - compute gradient on a small batch of examples (faster & parallelizable but noisy)
  - better scalability come at the cost of (sometimes) not converging to optimal point
  - increase robustness with larger batches at acceptable increased computational cost



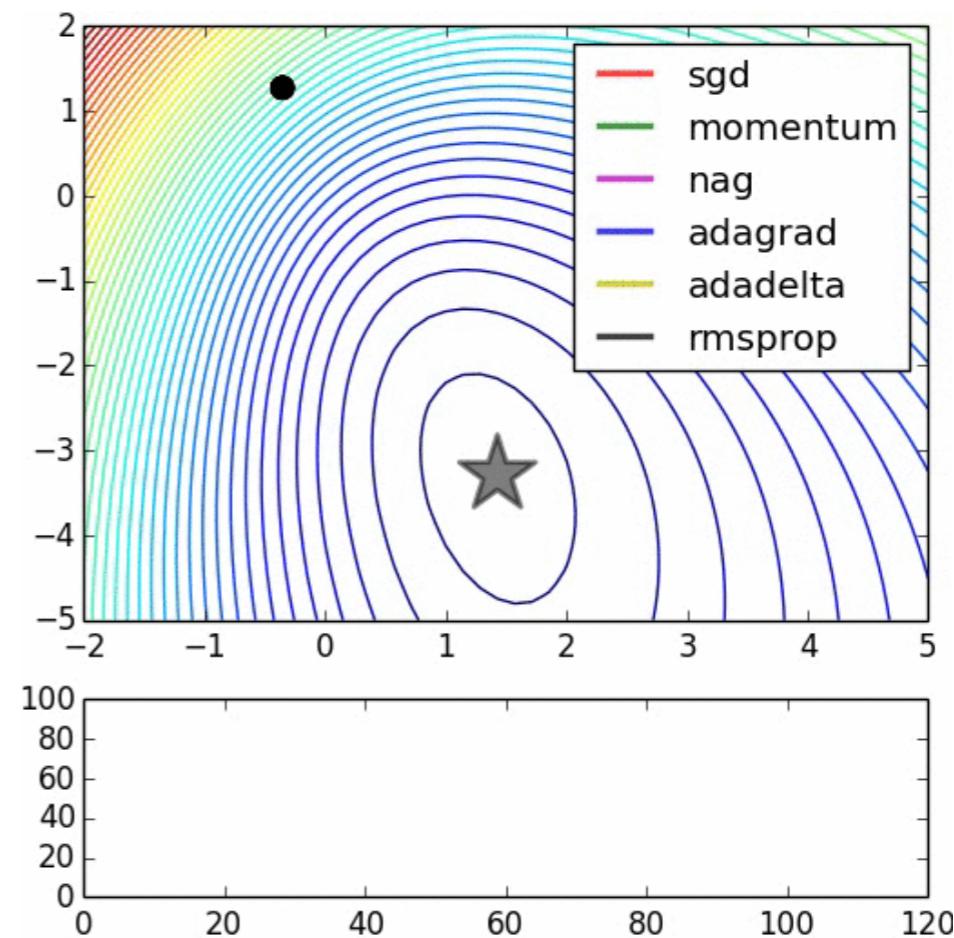
# Stochastic Gradient Descent

- Make the minimization more computationally efficient
  - compute gradient on a small batch of examples (faster & parallelizable but noisy)
  - better scalability come at the cost of (sometimes) not converging to optimal point
  - increase robustness with larger batches at acceptable increased computational cost
  - **many recipes exist to help convergence, by playing with the training setup**



# Stochastic Gradient Descent

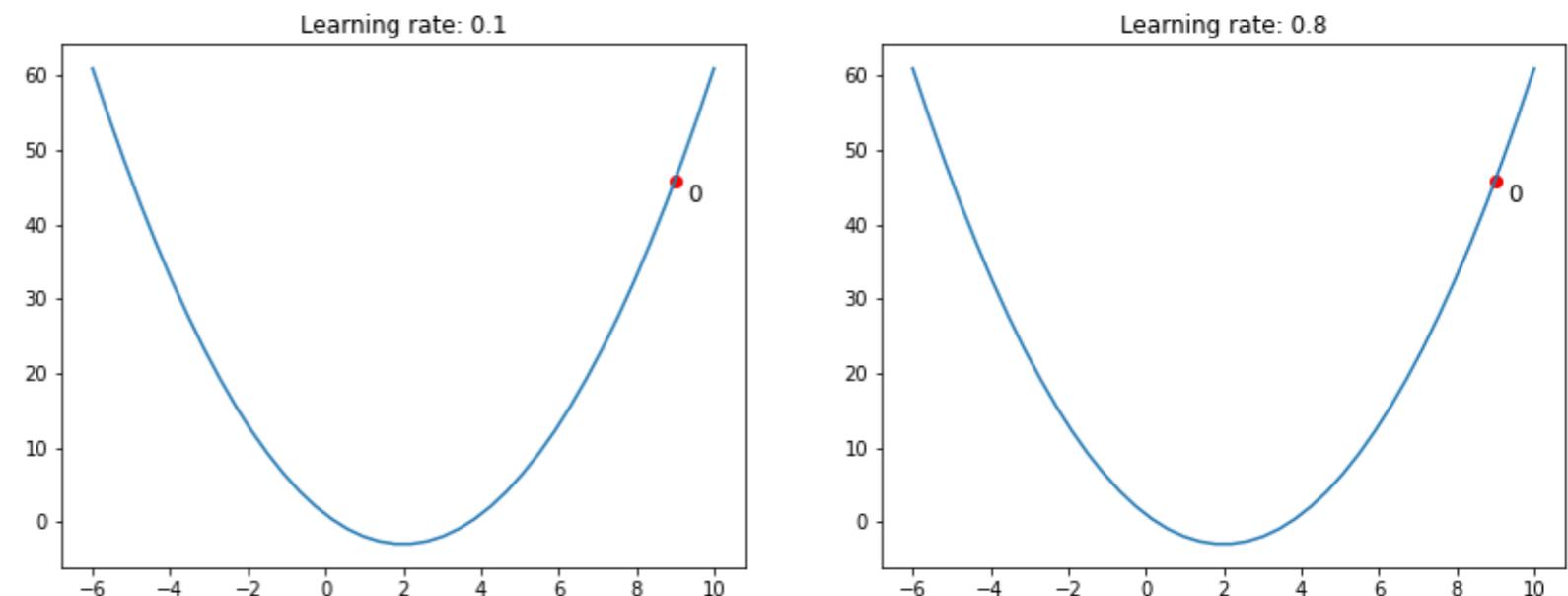
- Make the minimization more computationally efficient
  - compute gradient on a small batch of examples (faster & parallelizable but noisy)
  - better scalability come at the cost of (sometimes) not converging to optimal point
  - increase robustness with larger batches at acceptable increased computational cost
  - **many recipes exist to help convergence, by playing with the training setup**



# Stochastic Gradient Descent

- Make the minimization more computationally efficient
  - compute gradient on a small batch of examples (faster & parallelizable but noisy)
  - better scalability come at the cost of (sometimes) not converging to optimal point
  - increase robustness with larger batches at acceptable increased computational cost
  - **many recipes exist to help convergence, by playing with the training setup**

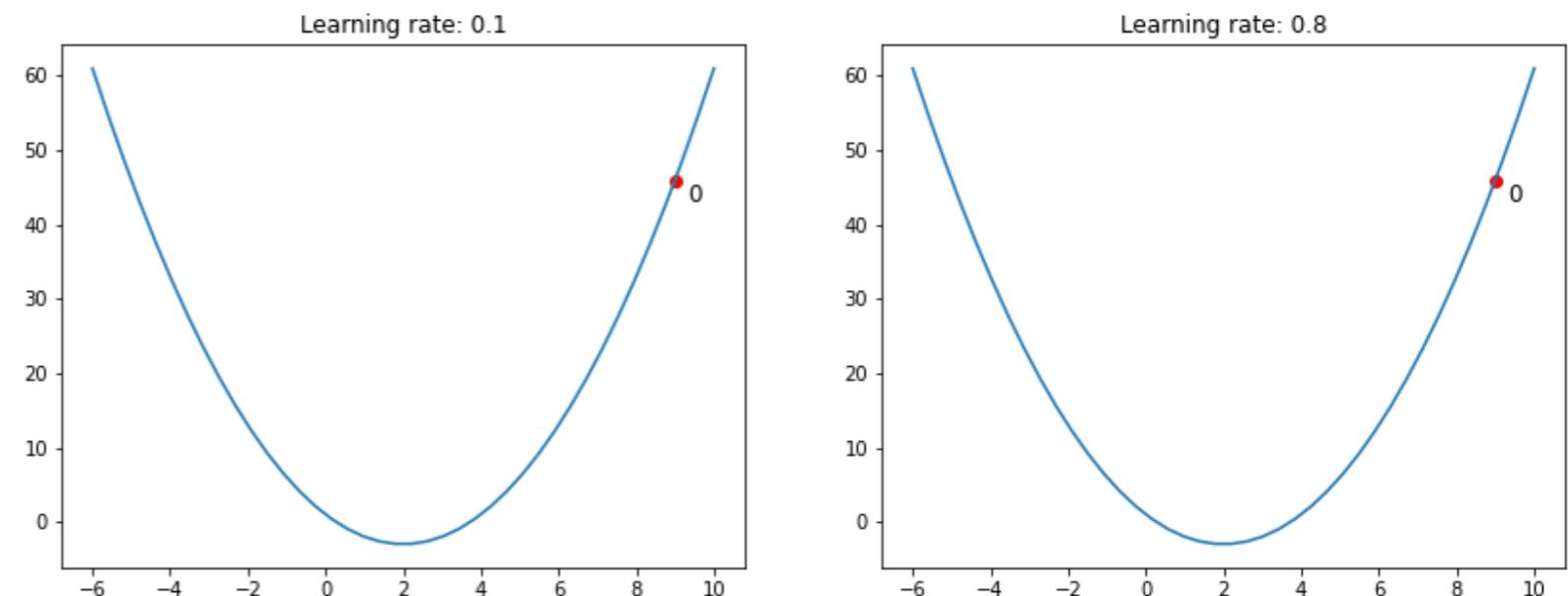
- Too small a learning rate, convergence very slow
- Too large a learning rate, algorithm diverges



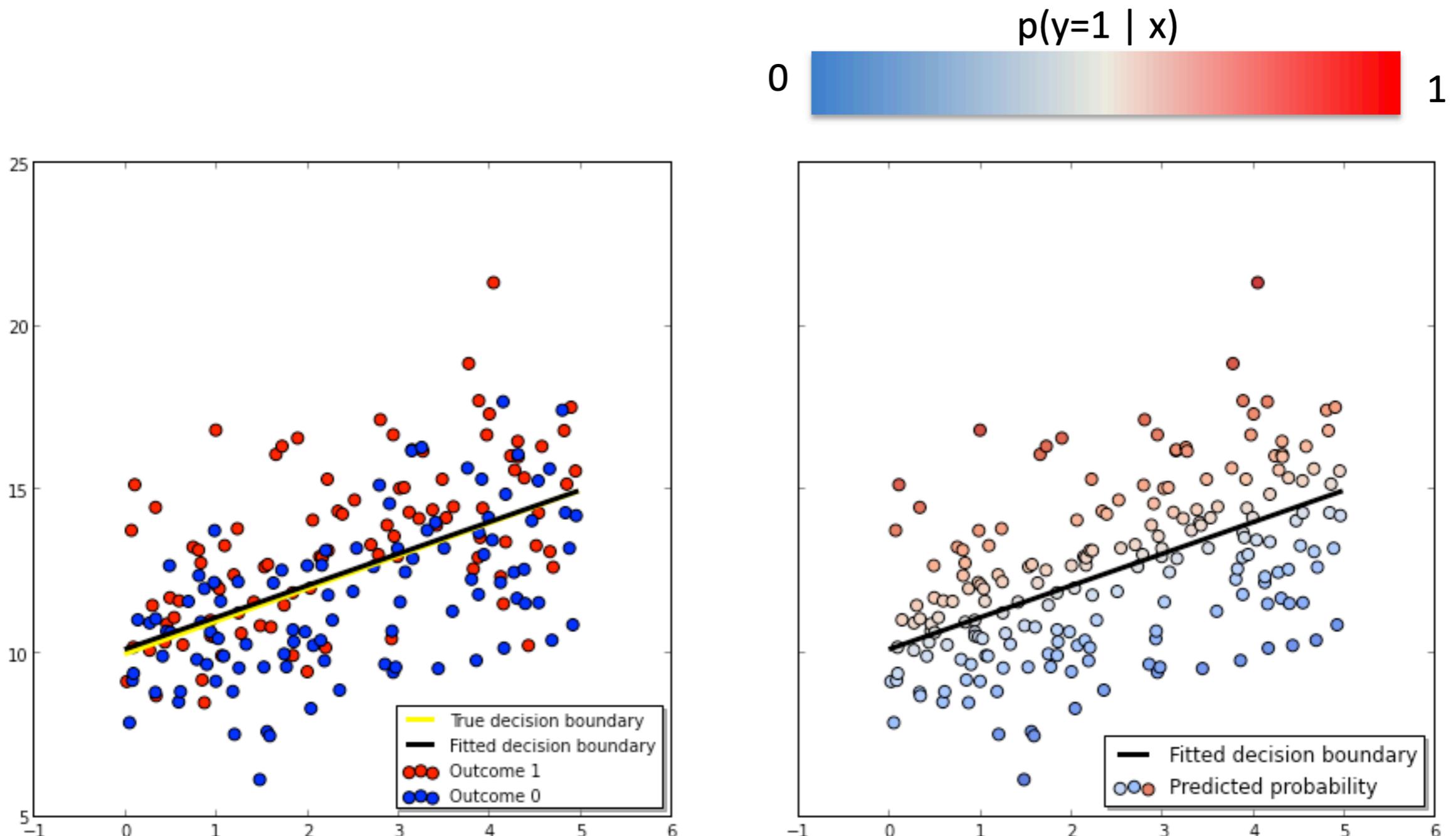
# Stochastic Gradient Descent

- Make the minimization more computationally efficient
  - compute gradient on a small batch of examples (faster & parallelizable but noisy)
  - better scalability come at the cost of (sometimes) not converging to optimal point
  - increase robustness with larger batches at acceptable increased computational cost
  - **many recipes exist to help convergence, by playing with the training setup**

- Too small a learning rate, convergence very slow
- Too large a learning rate, algorithm diverges



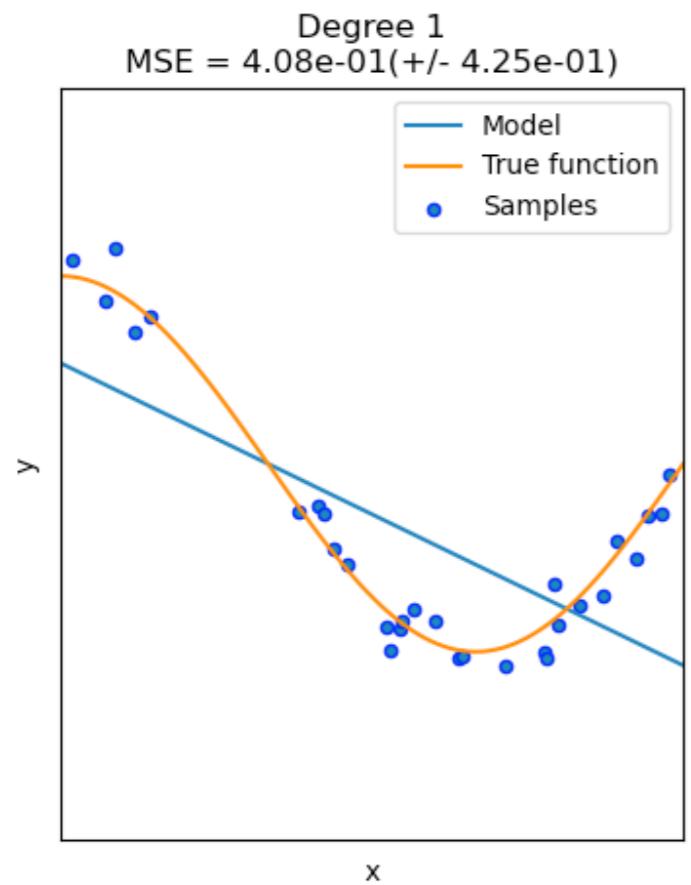
# Apply it to logistic regression



# Overfitting

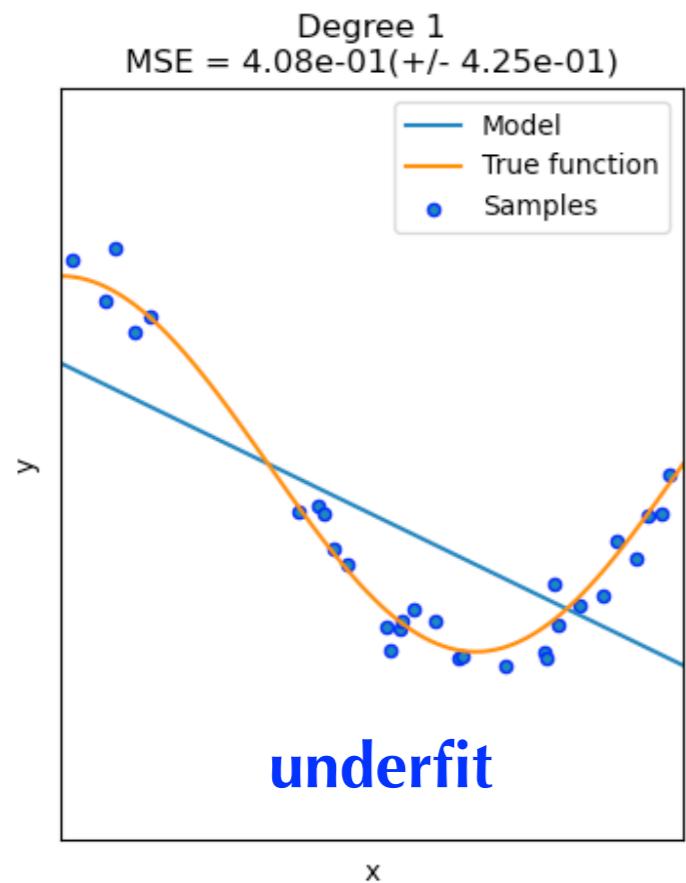
---

# Fooling the linear regression



- What if  $\mathbf{x}$  and  $\mathbf{y}$  are not linearly correlated?
- Can chose a basis functions  $\phi(\mathbf{x})$  to form new features such that  $\mathbf{y} = \mathbf{w}^T \phi(\mathbf{x})$ 
  - polynomial basis  $\phi(\mathbf{x}) = \{1, \mathbf{x}, \mathbf{x}^2, \mathbf{x}^3, \dots\}$
  - **Linear regression on new features  $\phi(\mathbf{x})$**
- **Problem: what basis functions to choose?** (e.g., up to which order polynomial)

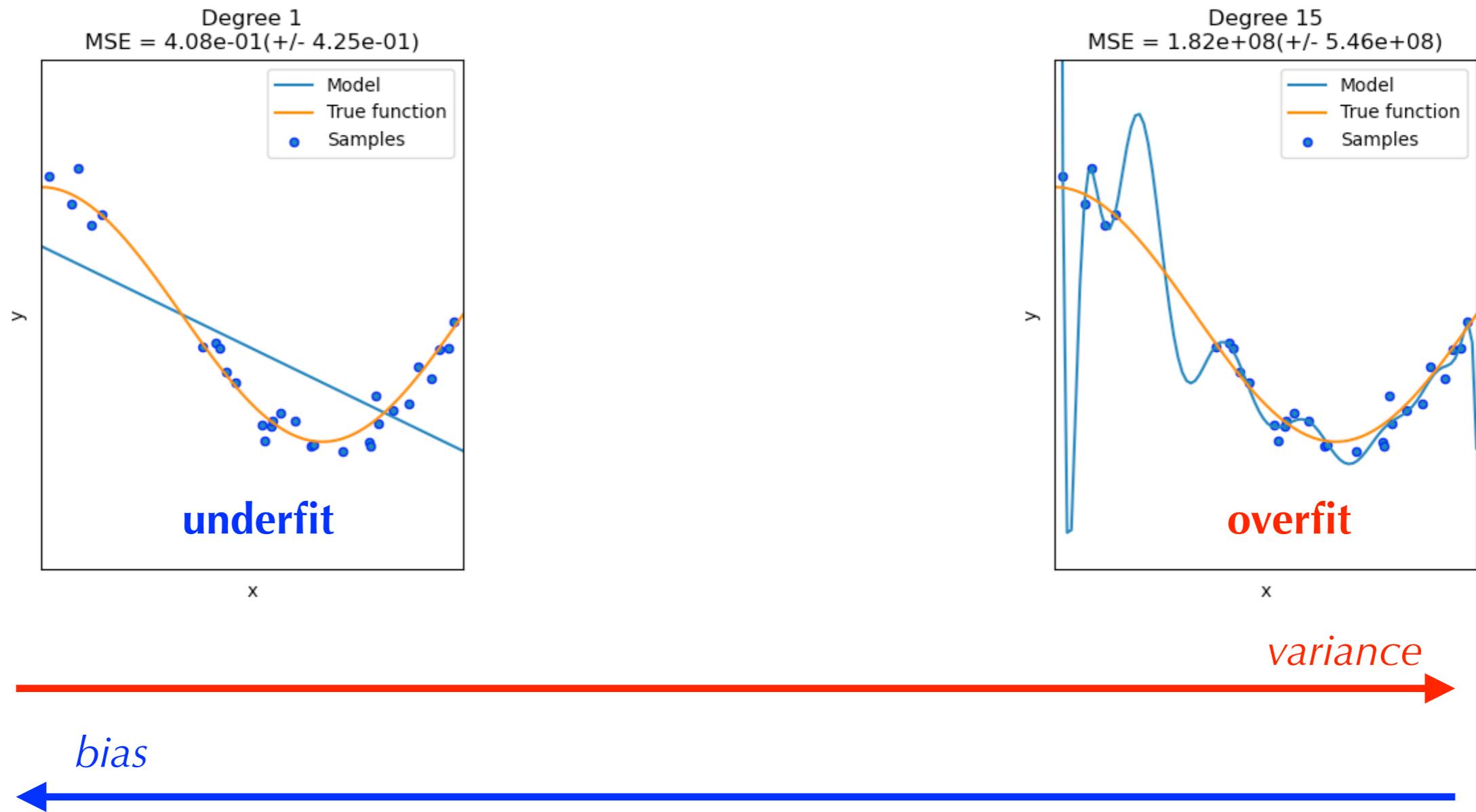
# Bias-variance tradeoff



*bias*

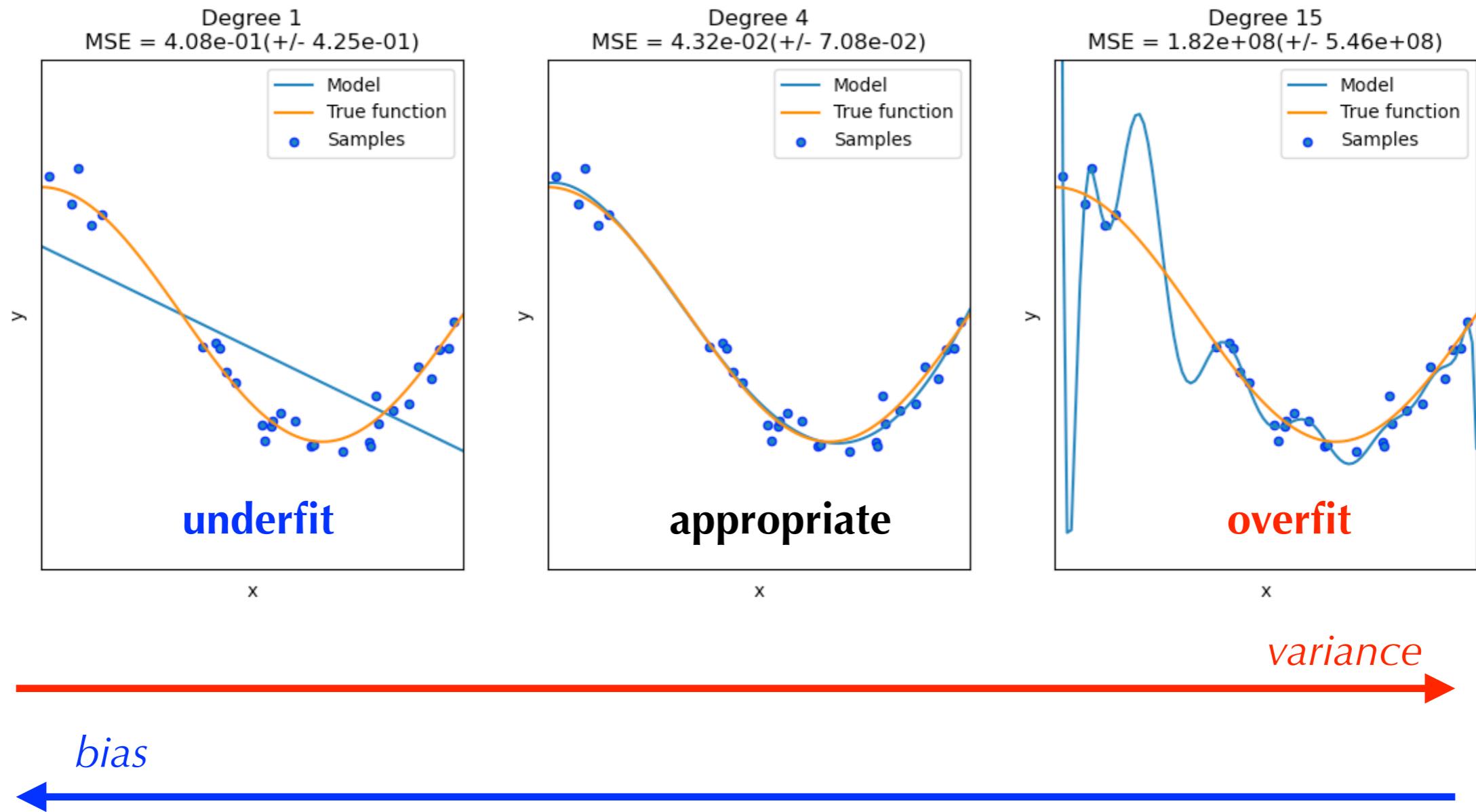
- **A too simple model underfits the data:** will deviate from data (high bias) but will not be influenced by peculiarities of the data (low variance) → not enough flexibility

# Bias-variance tradeoff



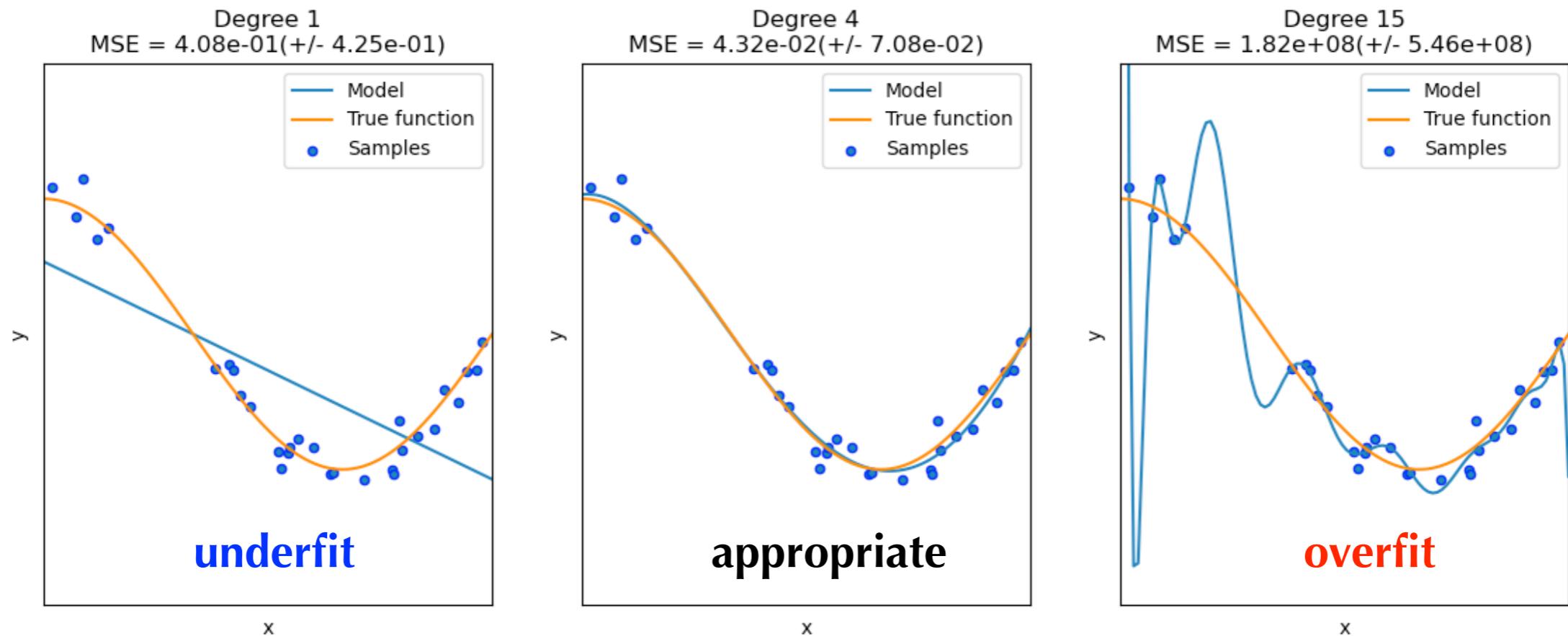
- **A too simple model underfits the data:** will deviate from data (high bias) but will not be influenced by peculiarities of the data (low variance) → not enough flexibility
- **A too complex model overfits the data:** will not deviate systematically from data (low bias) but will be very sensitive to noise in the data (high variance)

# Bias-variance tradeoff



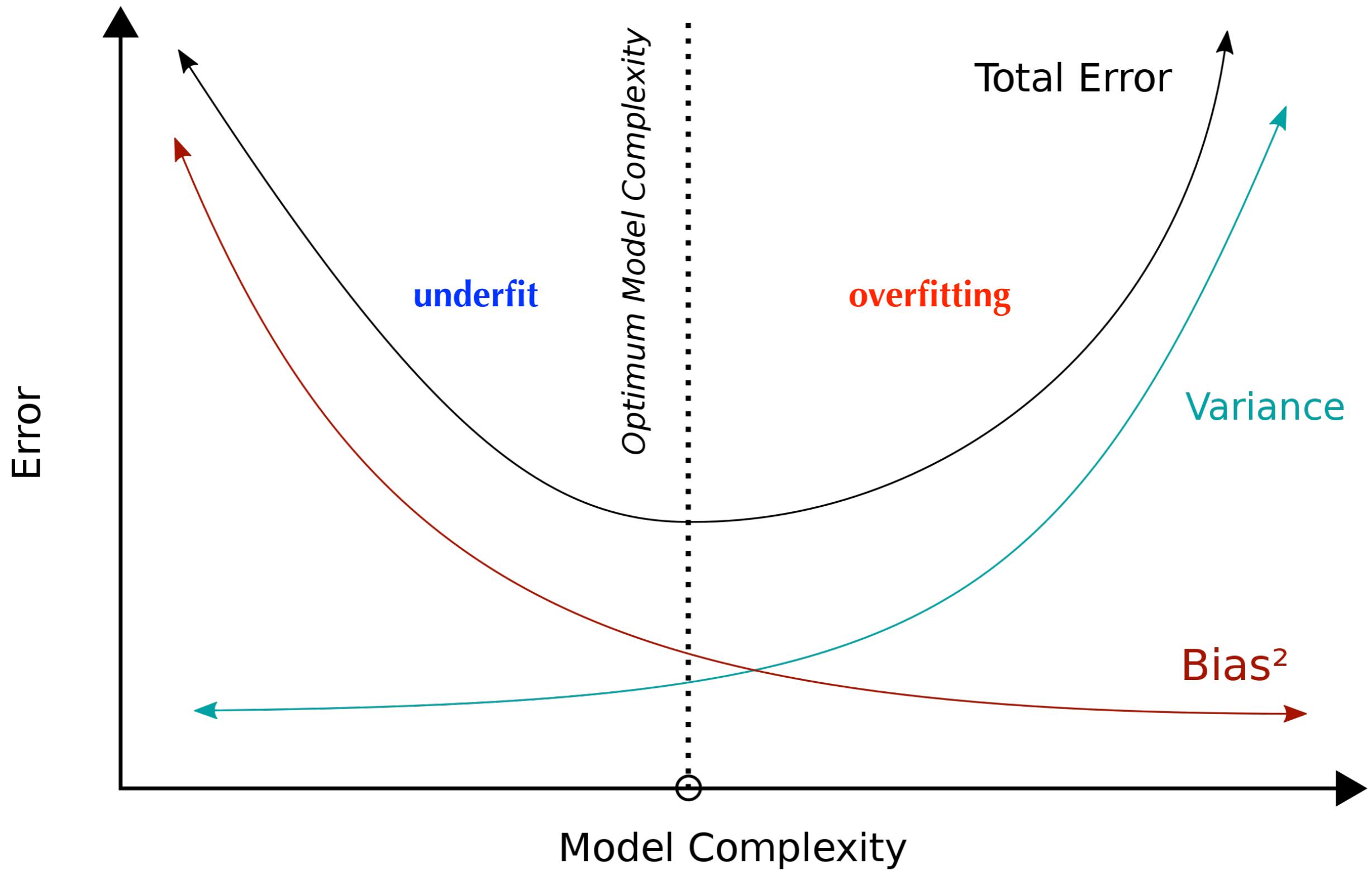
- **A too simple model underfits the data:** will deviate from data (high bias) but will not be influenced by peculiarities of the data (low variance) → not enough flexibility
- **A too complex model overfits the data:** will not deviate systematically from data (low bias) but will be very sensitive to noise in the data (high variance)

# Bias-variance tradeoff



- What models allow us to do is to **generalize** to unseen examples
- Different models can have better or worse generalization performance
- **Generalization error** = **systematic error** + **sensitivity to noise**
  - (bias)
  - (variance)

# Bias-variance tradeoff



# Regularization

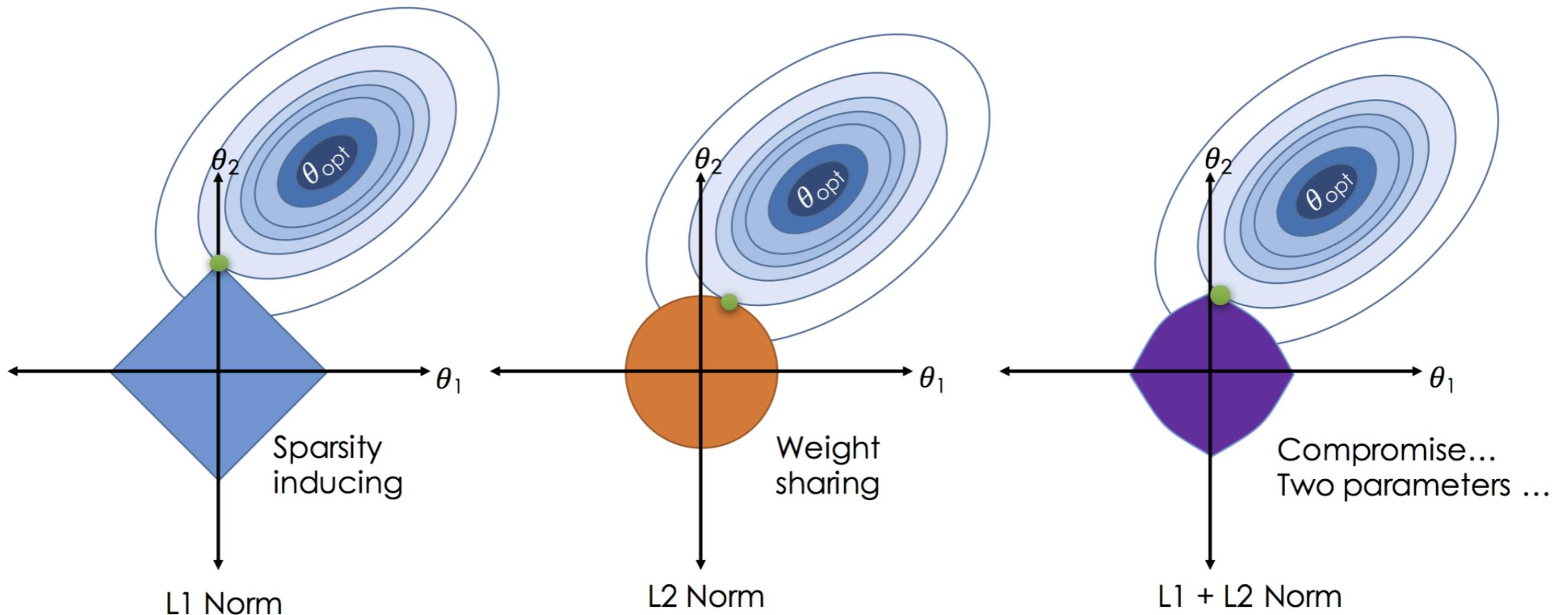
- A technique that helps avoid overfitting by shrinking the parameter estimates towards zero (i.e., discourages learning a more complex or flexible model)
- A modified loss is introduced, with a penalty term attached to each model parameter

$$\mathcal{L}_{reg} = \mathcal{L}(\mathbf{W}) + \lambda \Omega(\mathbf{W})$$

- Example:  $L_p$  regularization  $\Omega(\mathbf{W}) = \|\mathbf{W}\|^p = \sum_i |w_i|^p$
- The minimization is then a tradeoff between:

- pushing down the 1st term by taking advantage of the parameters
- pushing down the 2nd term by switching off the parameters
- The value of  $\lambda$  determines this tradeoff
  - $\lambda \sim 0$  : the penalty term has no effect
  - $\lambda \rightarrow +\infty$  : the impact of the penalty grows

# Regularizer types



- **L1 regularization:**  $\Omega(\mathbf{w}) = ||\mathbf{w}|| = \sum |w_i|$

- “Lasso regression” — enforce sparse weights

- **L2 regularization:**  $\Omega(\mathbf{w}) = ||\mathbf{w}||^2 = \sum w_i^2$

- “Ridge regression” — enforce small weights

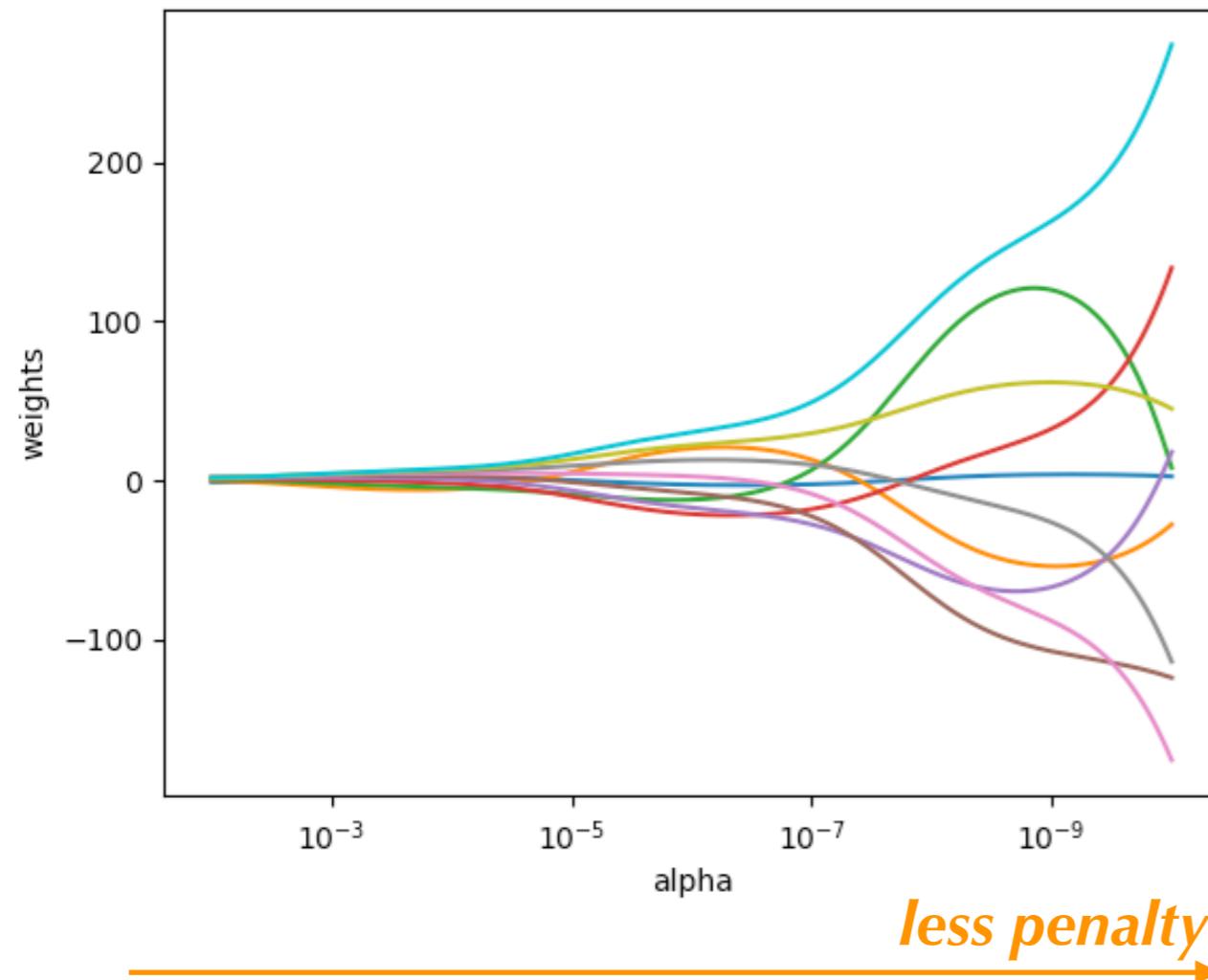
- **Elastic Net = L1+L2 regularization**

More about regularization  
[here](#) and [here](#)

# Regularizer types

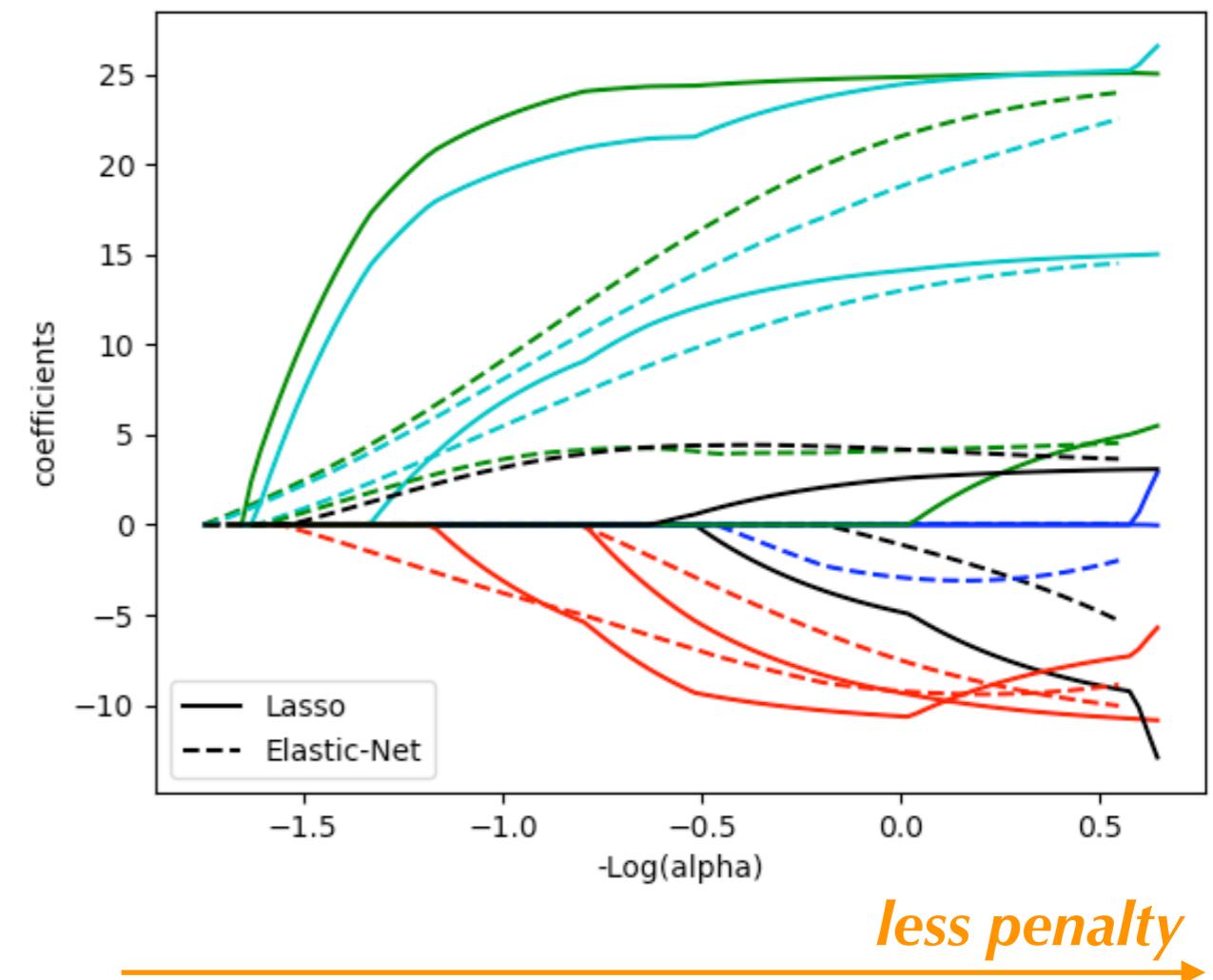
## L2 regularization

Ridge coefficients as a function of the regularization



## L1 + Elastic Net regularization

Lasso and Elastic-Net Paths

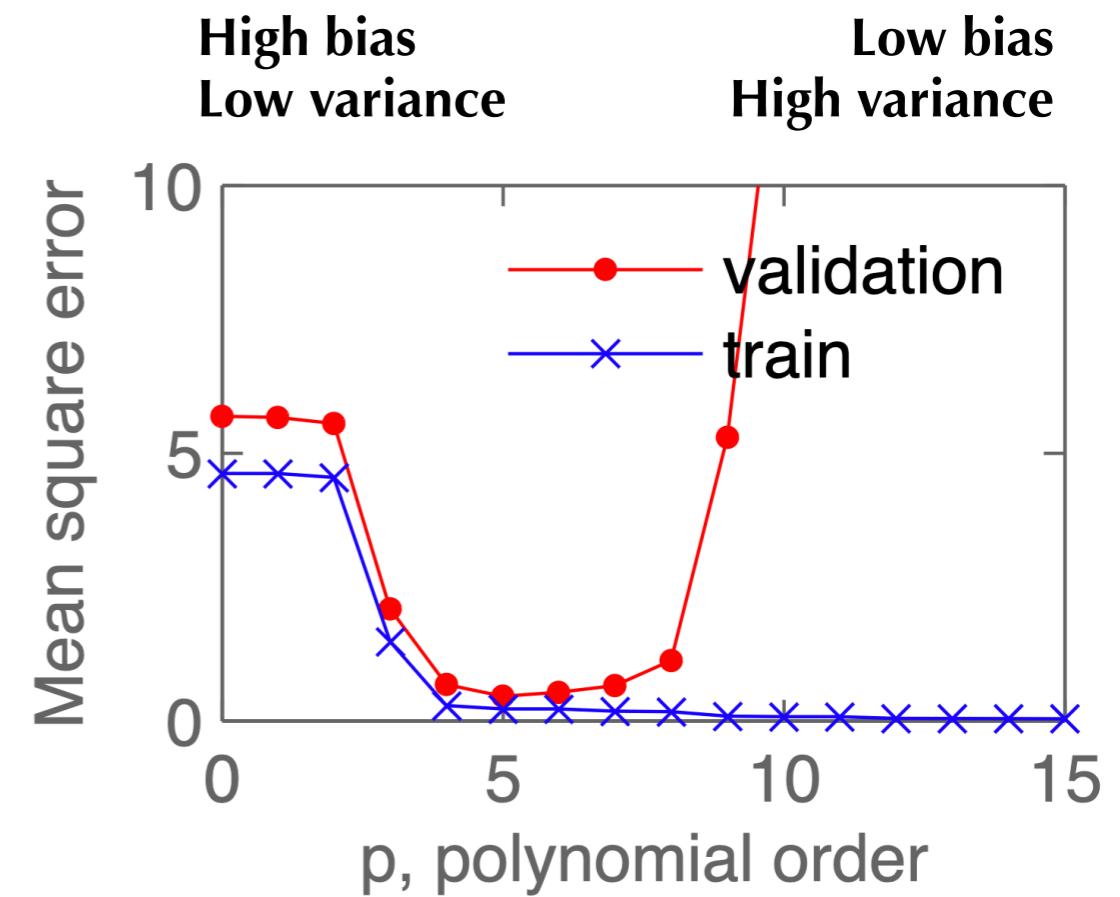
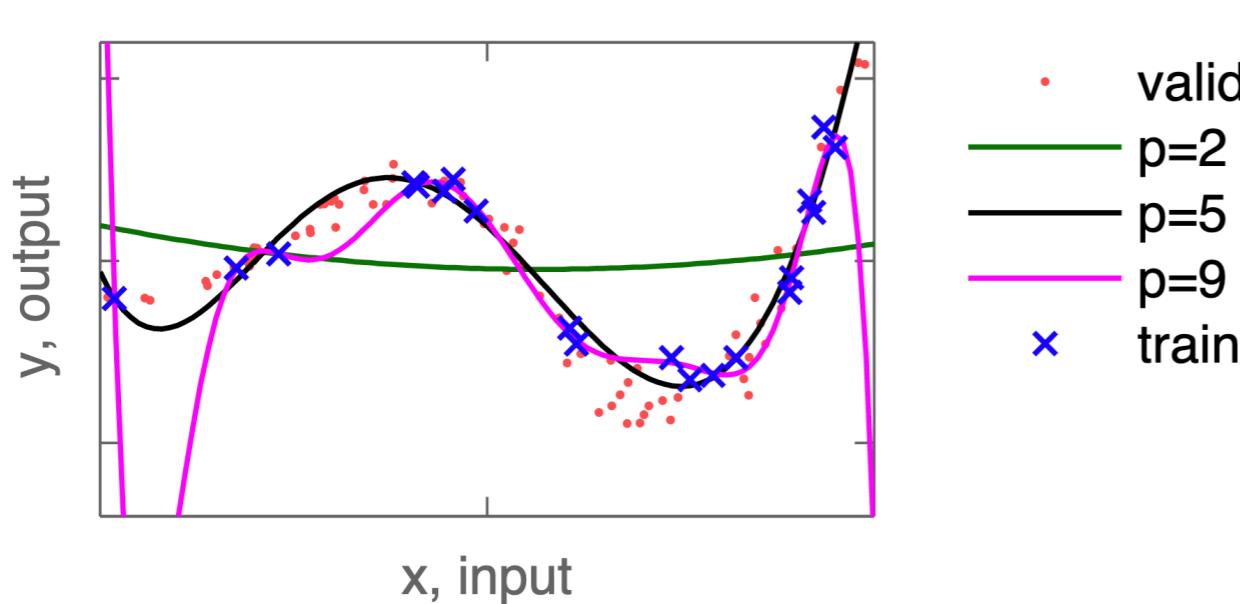


Some parameters switch on  
while being small

Not all parameters switch on

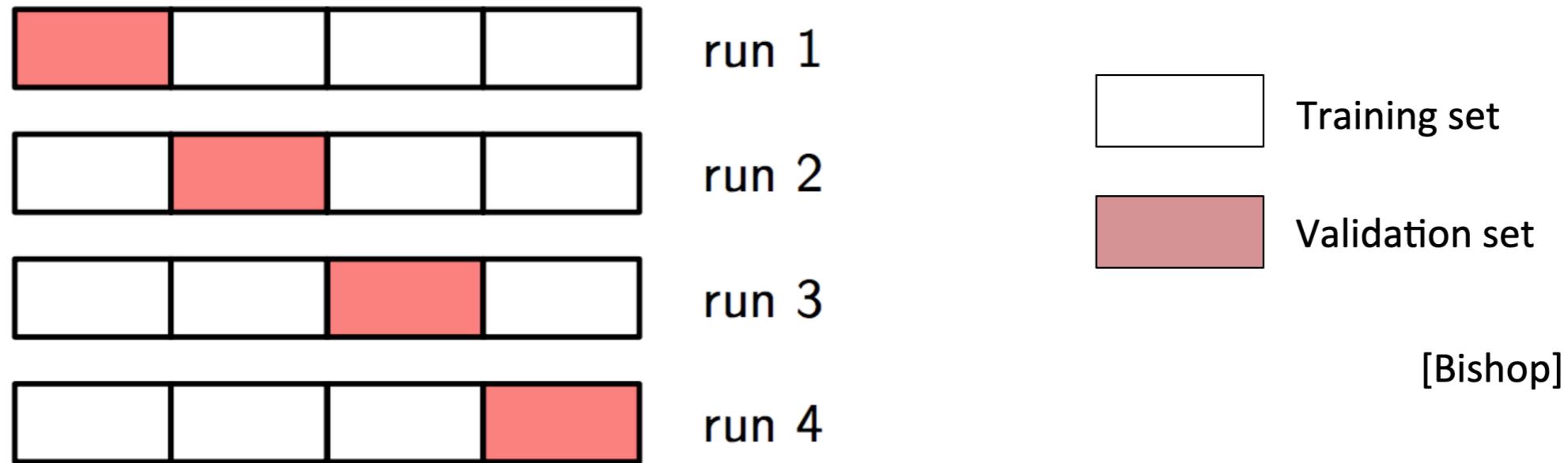
# How to monitor generalization?

- Split the sample in three parts:
  - **Training:** the biggest chunk — used to fit the model parameters
  - **Validation:** an auxiliary dataset to verify generalization — used to tune parameters
  - **Test:** the dataset for the final independent check after all parameters are fixed — needed since we tune performance on validation set during fitting



Example: 55% training, 20% validation, 25% test

# Cross validation to the rescue



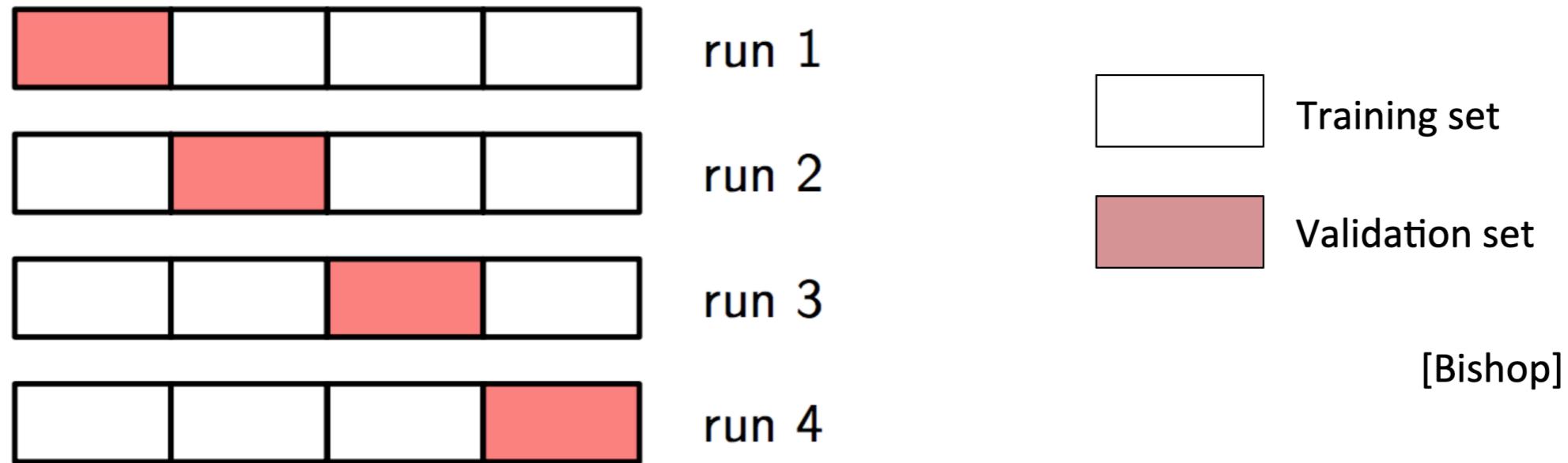
- **How does it work:**

- Randomly split the dataset in K chunks (called folds)
- Train on  $K-1$  chunks → obtain  $K-1$  losses
- Validate averaged  $K-1$  losses on the  $K$  chunk for optimization then iterate over K
- End up with K minimized losses → average them and test on full dataset

- **Allow you to use the full dataset (especially when it's small) while being unbiased**

- **Even when dataset not small, useful technique to estimate variance of expected performance**

# Choose number of folds



- **Too many folds:**

- **Small bias:** the estimator will be very accurate
- **Large variance:** due to small split sizes
- **Costly:** many experiments, large computational time

- **Too few folds:**

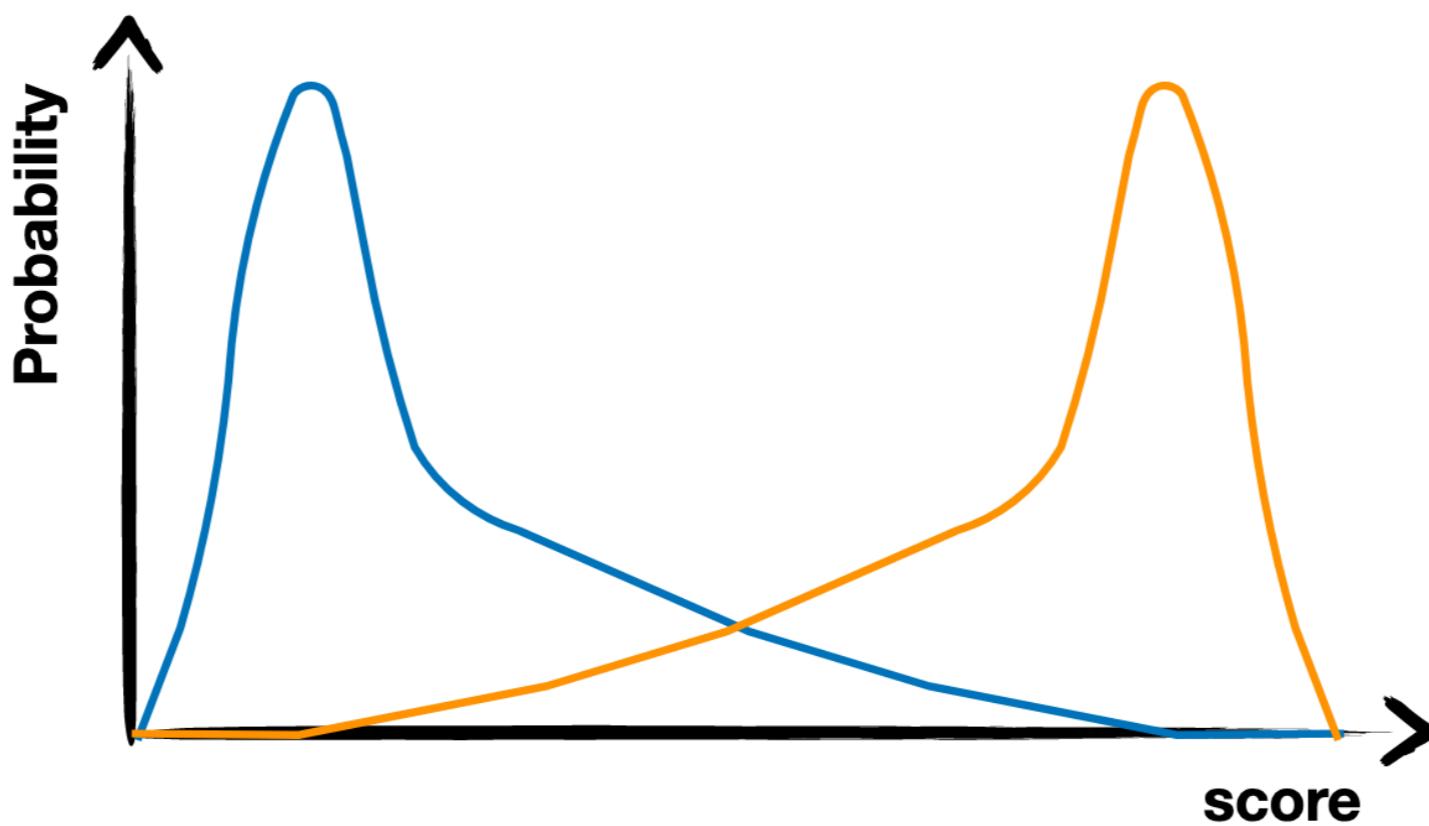
- **Cheap, computationally effective:** few experiments
- **Small variance:** average over many samples
- **Large bias:** estimated error rate conservative or smaller than the true error rate

# Performance Metrics

---

# Evaluate classifier performance

- Consider a binary classifier:  
its output  $\hat{y}$  is a number in the range  $\{0,1\}$ 
  - we call  $\hat{y}$  the classifier score
- If well trained, value should be close to 0 (1) for class-0 (class-1) examples



# Evaluate classifier performance

- Consider a binary classifier:  
its output  $\hat{y}$  is a number in the range  $\{0,1\}$

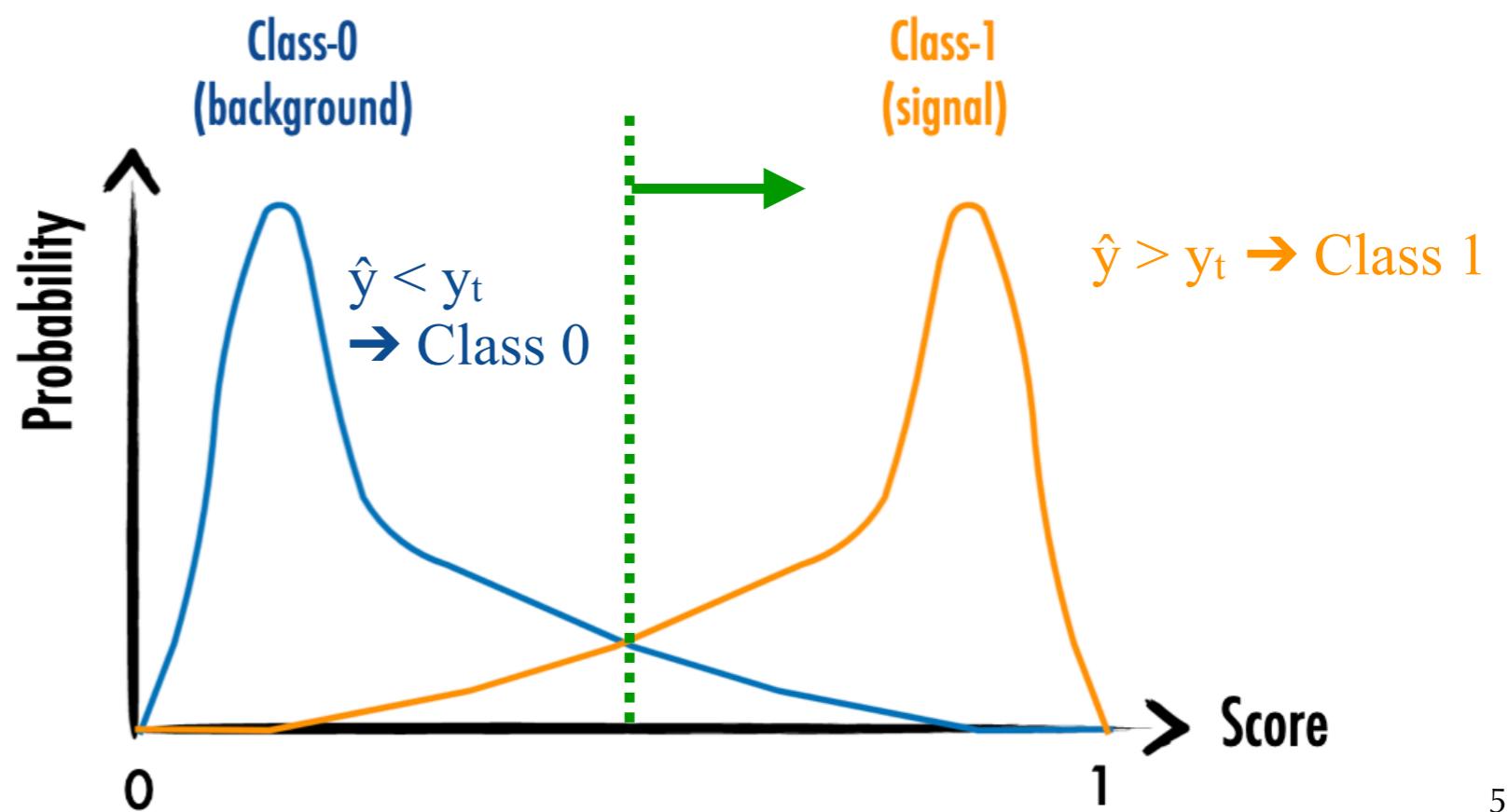
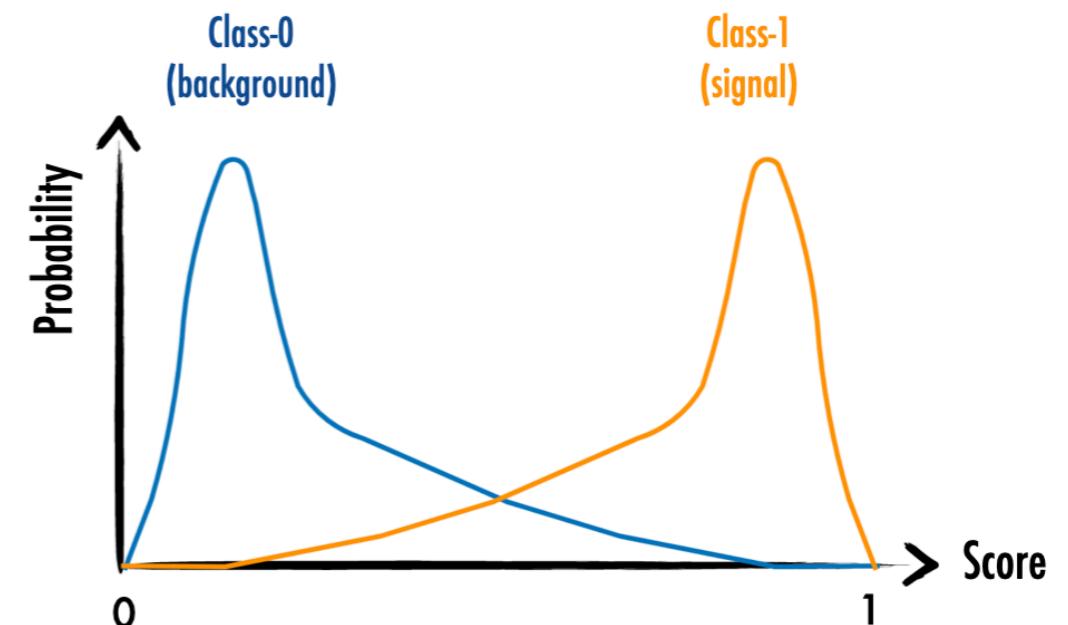
- we call  $\hat{y}$  the classifier score

- If well trained, value should be close to 0 (1) for class-0 (class-1) examples

- One usually defines a threshold  $y_t$  such that:

- $\hat{y} > y_t \rightarrow \text{Class 1}$

- $\hat{y} < y_t \rightarrow \text{Class 0}$



# Evaluate classifier performance

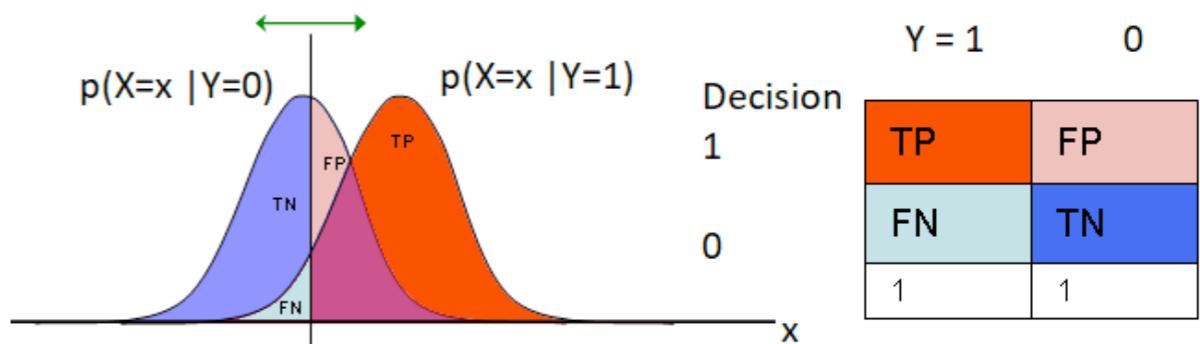
- Given the threshold  $y_t$  you can define:

		True classes		
		class 1	class 0	
Predicted classes	class 1	True Positive (TP)	False Positive (FP)	$\hat{y} > y_t$
	class 0	False Negative (FN)	True Negative (TN)	$\hat{y} < y_t$

- Rates are typically more informative:

$$\text{False Positive Rate (FPR)} = \frac{\mathbf{FP}}{\mathbf{FP} + \mathbf{TN}}$$

$$\text{True Positive Rate (FPR)} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FN}}$$

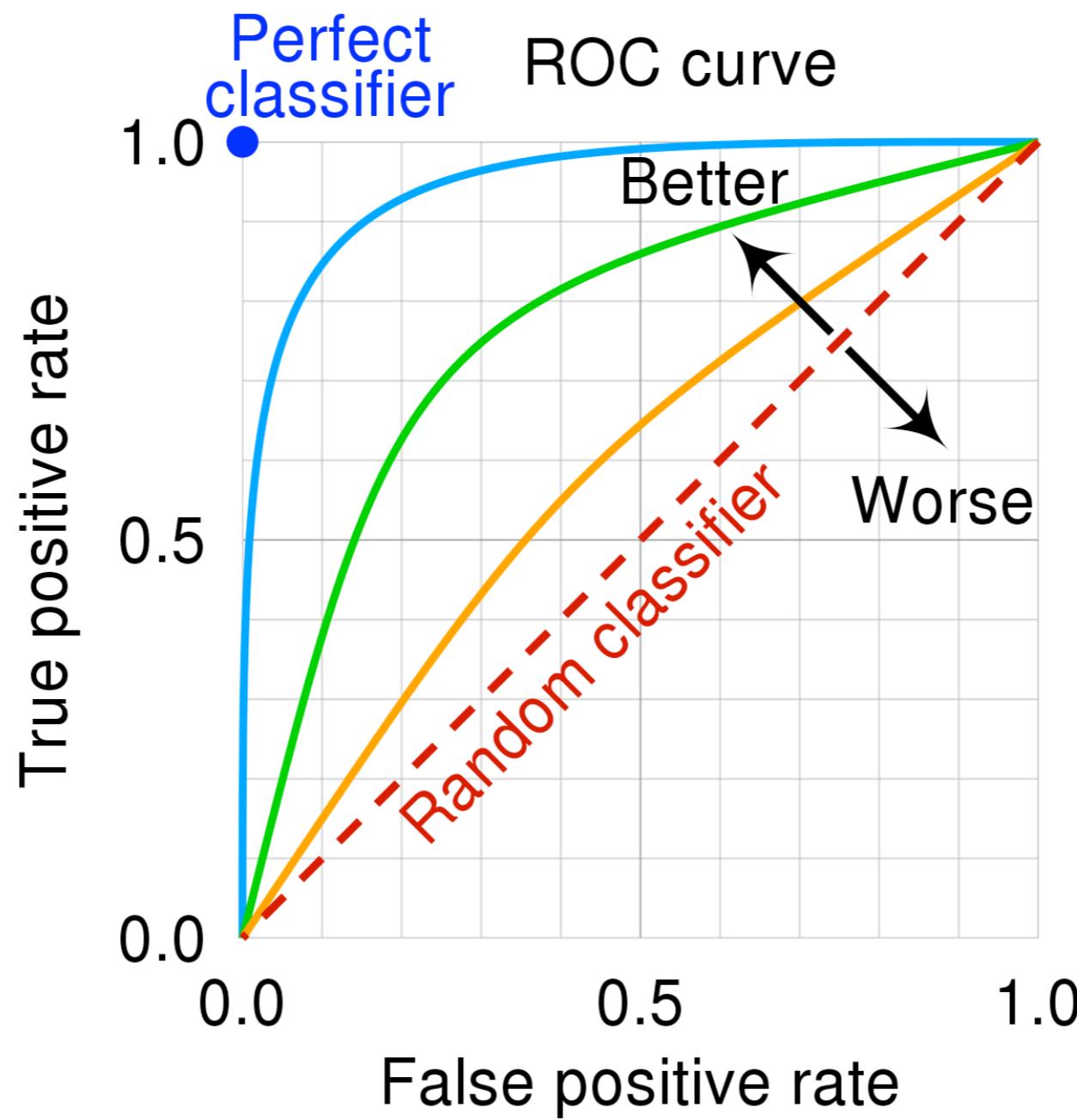


- And accuracy can be expressed as:

$$\frac{\mathbf{TP} + \mathbf{TN}}{\mathbf{TP} + \mathbf{FP} + \mathbf{FN} + \mathbf{TN}}$$

# Evaluate classifier performance

- You can plot the rates to obtain the so-called **Receiver Operating Curve (ROC)**
- The **Area Under The Curve (AUC)** can be used to tell how good your model is doing



# Summary lecture #1

- Machine learning uses mathematical and statistical models learned from data to characterize patterns and relations between inputs, and use this for inference / prediction
- Machine learning comes in many forms, much of which has probabilistic and statistical foundations and interpretations (i.e. *Statistical Machine Learning*)
- Discussed linear models today
- Tomorrow will be about nonlinear models → **Neural Networks!**
- This afternoon we'll go through **tools for data manipulation** — crucial to make data inputs to the ML tools we'll learn about in the next days