Machine learning

David Zarzoso
Aix Marseille Univ, CNRS, Centrale Med, M2P2 UMR 7340, Marseille

Introduction to machine learning

Data

The prediction problem

Linear regression

k-nearest neighbours

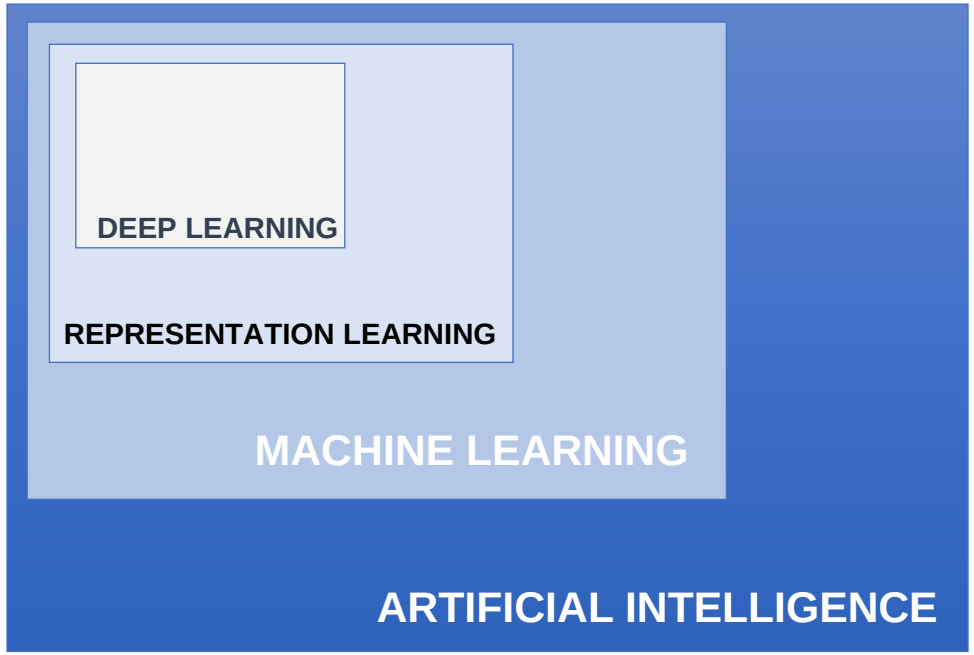
The curse of dimensionality

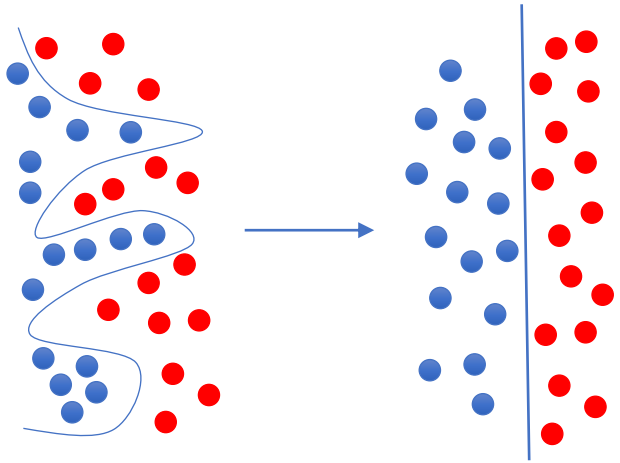
Gradient descent

Bibliography

- *The elements of statistical learning*, T. Hastie, R. Tibshirani and J. Friedman
- *Artificial Intelligence, a modern approach*, 3rd Edition, Stuart Russell and Peter Norvig
- *Deep learning*, I. Goodfellow, Y. Bengio and A. Courville
- Only for this first lecture → *Inaugural lecture at College de France,* Stephan Mallat (in French)
- Of course, many online courses are available nowadays, so do not hesitate to discover them! Some of them are
  - Lectures on *Learning from data*, Yaser Abu-Mostafa (Caltech)
  - MOOC *Machine Learning*, Andrew Ng (Stanford)

The history of machine learning in a nutshell





Can we imitate the human brain? Warren S. McCulloch and Walter Pitts (1943) → first model of artificial neurons with each neuron being either ON or OFF

Can computers be made to think? How to automate intellectual tasks usually performed by humans.

Symbolic AI → hardcode a sufficient number of explicit rules. Good start but not enough…

Can computers learn from data? How can a computer program its own codes.

Machine learning → Find statistical structure (patterns) by training on data (1990s).

Representation learning → The performance of the ML algorithms depend on the representation of the data. What is the most relevant information contained in the data?

The *deep* in deep learning computers can learn complex concepts building them out of simpler ones → Flatten boundaries to facilitate decisions

The birth of Artificial Intelligence (1956)

A PROPOSAL FOR THE

DARTMOUTH SUMMER RESEARCH PROJECT

ON ARTIFICIAL INTELLIGENCE

We propose that a 2 month, 10 man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.

J. McCarthy, Dartmouth College
M. L. Minsky, Harvard University
N. Rochester, I. B. M. Corporation
C. E. Shannon, Bell Telephone Laboratories

Everything started with…

DATA

Quantitative

Some measurements are bigger than others
Measurements close in value are close in nature
Examples: temperature in a room, speed of a car

Qualitative (or categorical)

Values belong to a discrete finite set, the elements of this set are called *classes*
No explicit ordering in the classes
Examples:

- Animal={cat, dog, bird, dolphin}
- Transportation={car, bus, train, plane}
- Digits={0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Four main problems in machine learning

SUPERVISED LEARNING

Given some INPUT DATA → Predict the OUTPUT DATA

- Quantitative data → Regression
- Categorical data → Classification

UNSUPERVISED LEARNING

Given some DATA → Extract information or learn something

- Finding similarities in data → Clustering
- Extracting the most relevant data → Learning representation

REINFORCEMENT LEARNING: trial and error to maximize a reward (autonomous driving, walking robot, play games…)

Some widely accepted notations for data

Input variable → X

Ex: time, distance to obstacle, city, colour, …

Output variable

- Quantitative Y. Ex: temperature, pressure, speed, angle, price
- Categorical G. Ex: animal, flower

p-dimensional variable: $X^T = (X_1, X_2, \dots, X_p)$, $Y^T = (Y_1, Y_2, \dots, Y_p)$

Ex: position, force, velocity, …

Observation → x, y or g

Ex: time=2s, distance to obstacle=10m, city=Houston, colour=red, …

For N observations, the i-th observation is x_i, y_i, g_i, with $i = 1, 2, \dots, N$. An observation can be a scalar or a vector

We can write in matrix form N observations of a p-dimensional variable X → $\boldsymbol{X} = \begin{pmatrix} x_{11} & \dots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{N1} & \dots & x_{Np} \end{pmatrix}$

Dimension →

Observation ↓

A first approach to the prediction problem

Estimate the output y knowing an input x

$x =$  What is this animal? → ALGORITHM → Estimation → $\hat{y}$

Learning: for known data, optimize the parameters of the algorithm so that $\hat{y} \approx y$

Generalization: if for unknown data x, the prediction is very close to reality, i.e. $\hat{y} \approx y$

→ based on some known mildness of the system (e.g. no discontinuities)

Formulation of the prediction problem

Imagine we have a dataset with N observations for the input variable X $\rightarrow \{x_1, x_2, \dots, x_N\}$ and N observations for the output variable Y $\rightarrow \{y_1, y_2, \dots, y_N\}$

> Ex: Y is the temperature of a component in an airfoil and X is the pressure and velocity of the fluid measured at different locations around that component.

Knowing this, we want to find a function f such that $Y = f(X)$, i.e. $y_i = f(x_i)$

This problem can be very difficult. We are not even sure that f exists. All we can do is find an estimate $\hat{f}$, such that $\hat{Y} = \hat{f}(X) \approx f(X) = Y$

We need a way to find $\hat{f}$, such that any error in the prediction $Y - \hat{f}(X)$ is penalized → Loss function $\mathcal{L}(Y, \hat{Y})$

$$\hat{f} = \underset{f}{\operatorname{argmin}}\, \mathcal{L}(Y, f(X)), \text{ restricted to the } \textit{training} \text{ dataset } \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

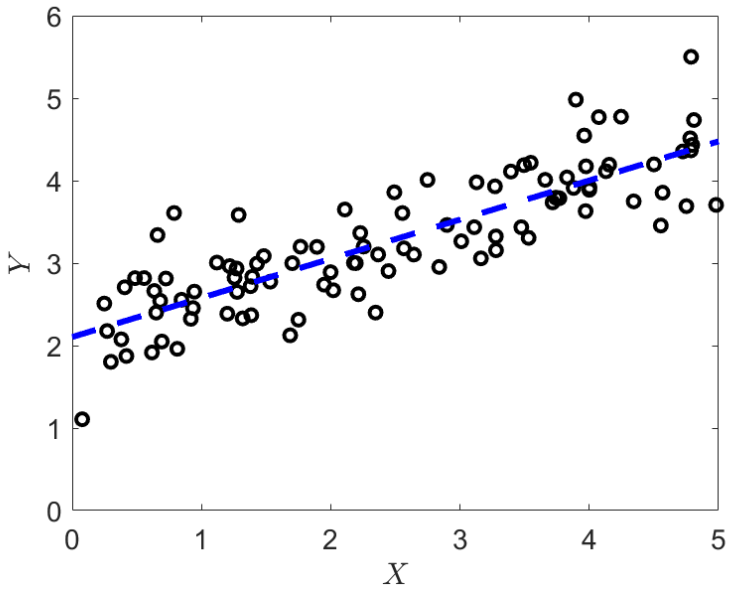
To solve this problem we need to (1) define the loss function $\mathcal{L}$ and (2) to make some assumptions on f

Exercise: linear regression as a prediction model

Let us assume without any loss of generality that we want to predict a real-valued output $Y \in \mathbb{R}$ and from a real-valued input $X \in \mathbb{R}$

The most common choice for the loss function is the *squared error loss* $\mathcal{L}(Y, \hat{Y}) = \|Y - \hat{Y}\|^2$

We try to find the function f making the assumption that it is linear, i.e. we can write $f(X) = \beta_0 + \beta_1 X$



Probabilistic approach to the prediction problem

So far we did not care about the real distribution of the training dataset

In general, we consider random variables with joint probability density $\mathcal{P}(X, Y)$

The best thing we can do is find the minimum of the expected prediction error

$$EPE(f) = \mathbb{E}\left[\mathcal{L}\left(Y, \hat{Y}\right)\right] = \mathbb{E}\left\|Y - \hat{Y}\right\|^2 = \iint \|y - f(x)\|^2 \mathcal{P}(x, y) dx dy$$

Knowing that $\mathcal{P}(X, Y) = \mathcal{P}(Y|X)\mathcal{P}(X)$, we can write $EPE(f) = \int \left[\underbrace{\int \|y - f(x)\|^2 \mathcal{P}(y|x) dy}_{= \mathbb{E}_{Y|X}[\|Y - f(X)\|^2|X]}\right] \mathcal{P}(x) dx$

$$= \mathbb{E}_X \underbrace{\mathbb{E}_{Y|X}[\|Y - f(X)\|^2|X]}_{> 0}$$

Probabilistic approach to the prediction problem

$$\min EPE(f) \Leftrightarrow \min \mathbb{E}[\|Y - f(X)\|^2 | X = x] \Rightarrow \hat{y} = \hat{f}(x) = \underset{a}{\operatorname{argmin}}\, \mathbb{E}[\|Y - a\|^2 | X = x]$$

We have to find a such that $\mathbb{E}[\|Y - a\|^2 | X = x]$ is minimum

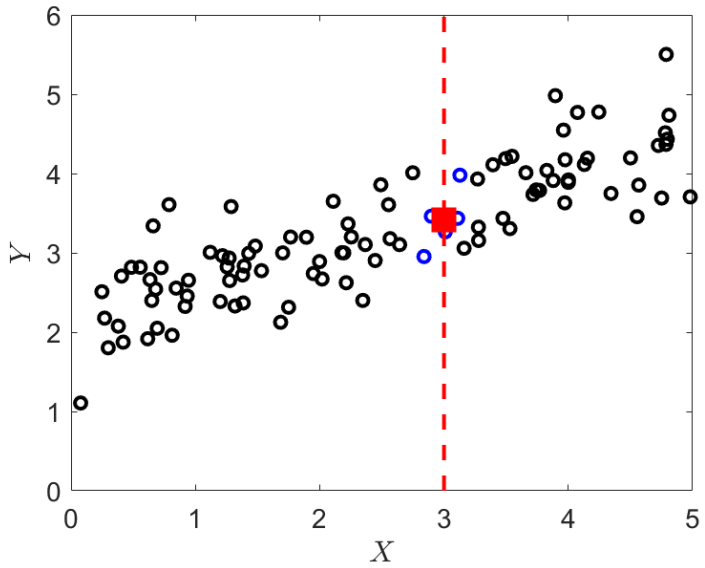
$$\mathbb{E}[\|Y - a\|^2 | X = x] = \mathbb{E}[\|Y\|^2 | X = x] + \|a\|^2 - 2a \cdot \mathbb{E}(Y | X = x) \xrightarrow{d/da} 2a\text{-}2\, \mathbb{E}(Y | X = x) = 0$$

The best choice for the estimate $\hat{f}$ is therefore the *conditional expectation of Y given $X = x$*, a.k.a. *regression function*

$$\hat{f}(x) = \mathbb{E}(Y | X = x)$$

<u>Remember:</u> all this applies under the assumption of a squared error loss function $\mathcal{L}(Y, \hat{Y}) = \left\|Y - \hat{Y}\right\|^2$
How we **estimate this regression function** is another problem…

k-nearest neighbour as a regression model



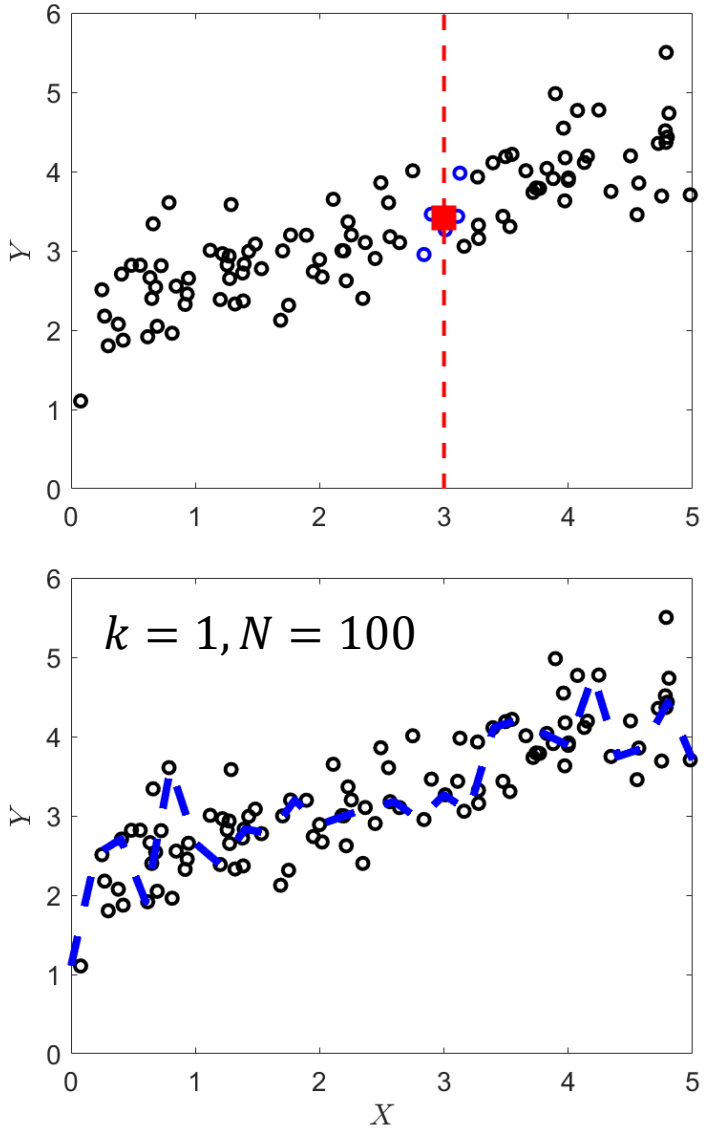
$$\hat{y} = \hat{f}(x) = \mathbb{E}(Y|X = x) \approx \frac{1}{k} \sum_{x_i \in \mathcal{N}_k(x)} y_i$$

Expectation is approximated by averaging over sample data

Conditioning at $X = x$ is approximated by conditioning on the neighborhood $\mathcal{N}_k(x)$ around x containing k elements

$\uparrow N \Rightarrow$ The points in $\mathcal{N}_k(x)$ get close to x
$\uparrow k \Rightarrow$ The average is more stable
$N, k \to \infty$ and $k/N \to 0 \Rightarrow$ The average converges to the conditional expectation

k-nearest neighbour as a regression model



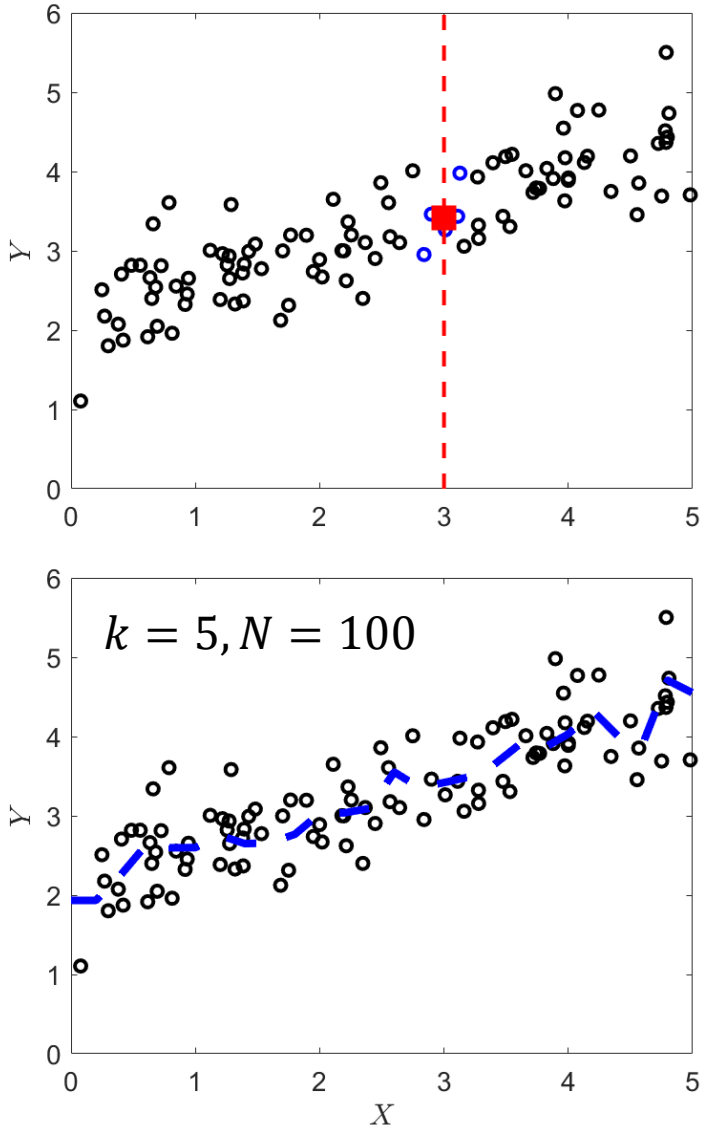
$$\hat{y} = \hat{f}(x) = \mathbb{E}(Y|X = x) \approx \frac{1}{k} \sum_{x_i \in \mathcal{N}_k(x)} y_i$$

Expectation is approximated by averaging over sample data

Conditioning at $X = x$ is approximated by conditioning on the neighborhood $\mathcal{N}_k(x)$ around x containing k elements

$\uparrow N \Rightarrow$ The points in $\mathcal{N}_k(x)$ get close to x
$\uparrow k \Rightarrow$ The average is more stable
$N, k \to \infty$ and $k/N \to 0 \Rightarrow$ The average converges to the conditional expectation

k-nearest neighbour as a regression model



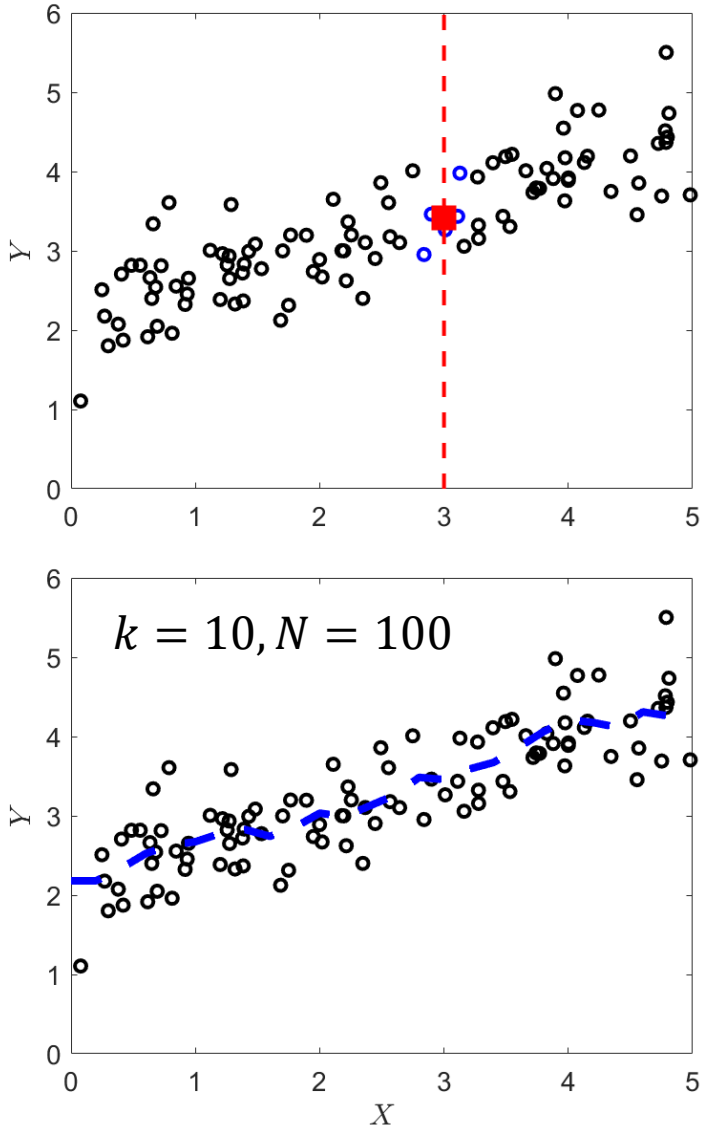
$$\hat{y} = \hat{f}(x) = \mathbb{E}(Y|X = x) \approx \frac{1}{k} \sum_{x_i \in \mathcal{N}_k(x)} y_i$$

Expectation is approximated by averaging over sample data

Conditioning at $X = x$ is approximated by conditioning on the neighborhood $\mathcal{N}_k(x)$ around x containing k elements

$\uparrow N \Rightarrow$ The points in $\mathcal{N}_k(x)$ get close to x
$\uparrow k \Rightarrow$ The average is more stable
$N, k \to \infty$ and $k/N \to 0 \Rightarrow$ The average converges to the conditional expectation

k-nearest neighbour as a regression model



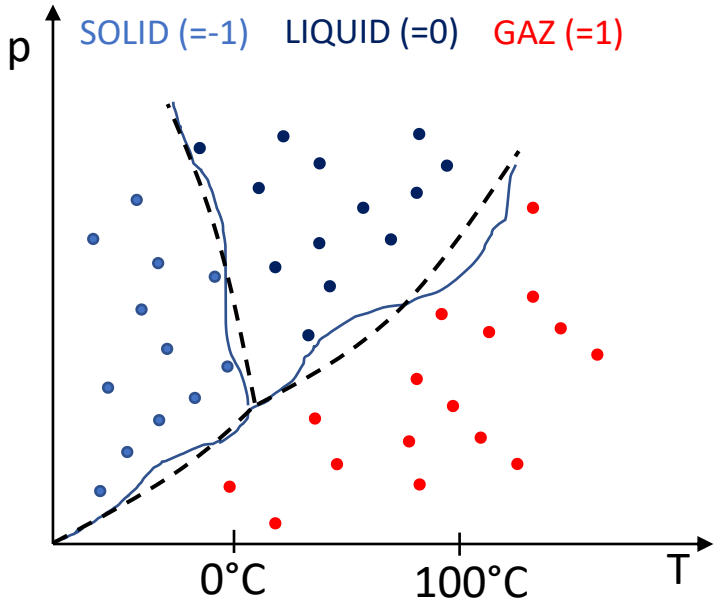
$$\hat{y} = \hat{f}(x) = \mathbb{E}(Y|X = x) \approx \frac{1}{k} \sum_{x_i \in \mathcal{N}_k(x)} y_i$$

Expectation is approximated by averaging over sample data

Conditioning at $X = x$ is approximated by conditioning on the neighborhood $\mathcal{N}_k(x)$ around x containing k elements

$\uparrow N \Rightarrow$ The points in $\mathcal{N}_k(x)$ get close to x
$\uparrow k \Rightarrow$ The average is more stable
$N, k \to \infty$ and $k/N \to 0 \Rightarrow$ The average converges to the conditional expectation

k-nearest neighbour as a classification model



k-nearest neihbour seems to allow some generalization because the true boundary is close to the real one

$$\hat{y} = \hat{f}(x) = \mathbb{E}(Y|X = x) \approx \frac{1}{k} \sum_{x_i \in \mathcal{N}_k(x)} y_i \Rightarrow \begin{cases} \text{SOLID}, if\ -1 \leq \hat{y} < -0.5 \\ \text{LIQUID}, if\ -0.5 \leq \hat{y} \leq 0.5 \\ \text{GAZ}, if\ 0.5 < \hat{y} \leq 1 \end{cases}$$

Expectation is approximated by averaging over sample data

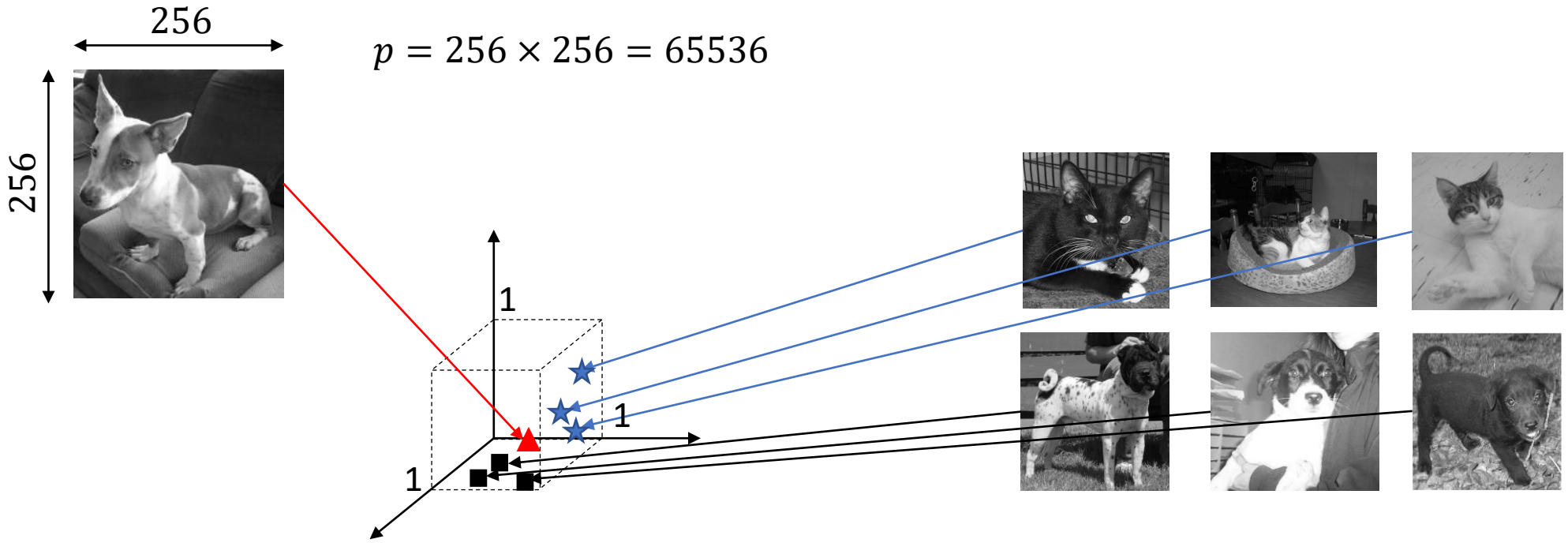
Conditioning at $X = x$ is approximated by conditioning on the neighborhood $\mathcal{N}_k(x)$ around x containing k elements

$\uparrow N \Rightarrow$ The points in $\mathcal{N}_k(x)$ get close to x
$\uparrow k \Rightarrow$ The average is more stable
$N, k \to \infty$ and $k/N \to 0 \Rightarrow$ The average converges to the conditional expectation

So basically, one single parameter (k) to fit and for large traning datasets ($N \to \infty$) the average converges to the conditional expectation → We found a universal approximator! **WRONG**

(1) In practice, we do not have $N \to \infty$
(2) Number of effective parameters to fit $\sim N/k$
(3) For high dimensions (large p) to capture a few percentage of the training data to compute the average we need to cover a large percentage of the range of each input variable → Curse of dimensionality

The curse of dimensionality



To capture a fraction r of the training dataset (in the unit hyper-cube), we need to consider a hyper-cube of edge of length $e_p = r^{1/p}$. Therefore, for $p = 65536$, if we want to use 1% of the training dataset to compute *knn*, we need to cover a hyper-cube of edge $e_p = 0.999929733$ …, i.e. almost the whole unit hyper-cube → In high-dimensionality there are no *nearest* neighbours → **The neighborhoods are no longer local**

Machine learning is all about optimization

$\hat{f} = \underset{f}{\operatorname{argmin}}\, \mathcal{L}(Y, f(X))$, restricted to the *training* dataset $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$

Imagine that f is a function that can be parametrized, i.e. $f(X) \equiv f(X; \theta_1, \theta_2, \dots, \theta_n) = f(X; \boldsymbol{\theta})$

The problem reduces to finding $\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathcal{L}(Y, f(X; \boldsymbol{\theta}))$

Solving this set of equations is most of the time not tractable analytically and we need to use iterative methods

Here we introduce the **gradient descent** method, which works **only for differentiable and convex functions**

$$\boldsymbol{\theta}_{n+1} \leftarrow \boldsymbol{\theta}_n - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$$

Assignment — Learning rate

η small → the algorithm converges slowly to the solution
η large → the algorithm jumps around the solution or even diverges

How to compute $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$ and update the parameters?

Three types of gradient descent

Stochastic gradient descent (SGD)

- The gradient is approximated by the gradient at a given sample of the training dataset
- The parameters are updated at every sample
- ☺ Low memory requirements, faster initial progress (update parameters much more frequently), can escape local minima
- ☹ Slow convergence

Full-batch gradient descent (aka GD)

- The gradient is approximated by the averaged gradient over the whole training dataset
- The parameters are updated once using all the dataset
- ☺ Can converge faster than SGD
- ☹ Large memory requirements

Mini-batch gradient descent (a combination of the previous ones, but ☹ tuning of the mini-batch size required)

- The training dataset is randomly split into subsets, called *mini-batches* or simply *batches*
- For each batch:
  - The gradient is approximated by the averaged gradient over the samples of the batch
  - The parameters are updated

N.B: An *epoch* is one iteration over the whole training dataset