# Introduction to Machine Learning

Nicolas Keriven
CNRS, IRISA, Rennes

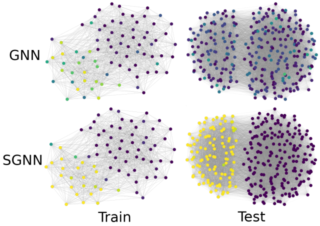
(material from Florent Chatelain, Olivier Michel)

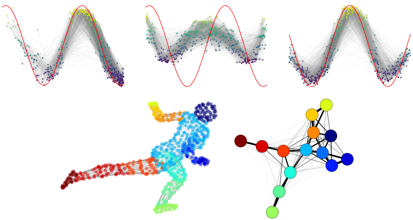
ENSTA 2023

# The presenter

## Nicolas Keriven

- 2010-2014 : Ecole polytechnique (X10), Master MVA
- 2014-2017 : Ph.D. at Inria Rennes
- 2017-2019 : Post-doc at ENS Ulm
- 2019 : Chargé de Recherche CNRS at Gipsa-lab, Grenoble
- 2023 : join IRISA, Rennes
- Research interest : ML theory, dimensionality reduction, Graph ML, Graph Neural Networks



## Organization

- $8 \times 3h$ lecture and practices sessions

| | Wednesday | Thursday | Friday | Monday | Tuesday |
|---|---|---|---|---|---|
| 09h30-12h45 | Intro | | SVM | Introduction to deep learning | Adv. Deep Learning + Exam. QCM |
| **Lunch** | | | | | |
| 14h15-17h30 | Discr. analysis + PCA | Linear models + Regul. | Clustering | Time series, RL and RNN | |

### Objectives

- Understand the theoretical basis of data science/machine learning/AI
- Implement/apply data science algorithms and models using state-of-the-art frameworks
- Learn how to learn : (of course) not exhaustive, and somewhat subjective !

# The material

- Slides (pdf) and notebooks available here : `https://github.com/nkeriven/ensta-mt12`
- Jupyter notebooks are available to illustrate concepts and methods in Python (.ipynb files)
- Notebooks can be run remotely on Google Colab

Part of this material is from previous years' presenters, Florent Chatelain (McF, GIPSA-lab) and Olivier Michel (CNRS, GIPSA-lab).

# Additional material

## Reference books

- Trevor Hastie, Robert Tibshirani et Jerome Friedman (2009) The Elements of Statistical Learning (2nd Edition) *Springer Series in Statistics*
- Christopher M. Bishop (2007) Pattern Recognition and Machine Learning *Springer*
- Kevin P. Murphy (2012) Machine Learning. A Probabilistic Perspective *MIT Press*

## Supplementary materials, datasets, online courses, ...

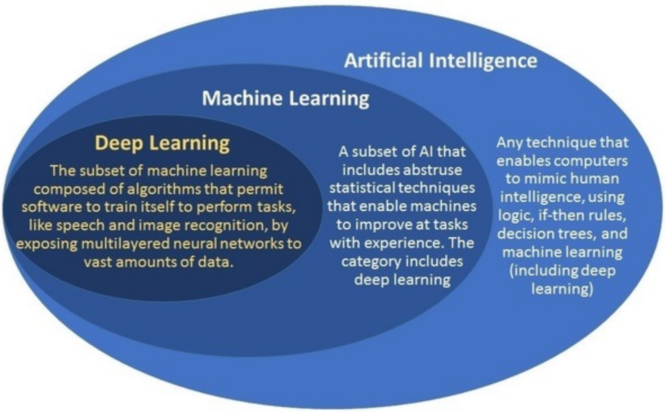
- `https://www.coursera.org/course/ml` *very popular MOOC (Andrew Ng)*
- `https://work.caltech.edu/telecourse.html` *more involved MOOC (Y. Abu-Mostafa)*
- `https://scikit-learn.org/stable/auto_examples/index.html`
  Examples from the sklearn library

Many (many) more material online, learn how to find it, interpret it, use it !
**Always with a grain of salt**

# Table of Contents

## Machine Learning $\subset$ Artificial Intelligence

A (simplified) picture...



## Data Science Objective

A (simplified !) picture...

**Objective of ML/Data Science**

- How to extract knowledge or insights from data ?
- How to use it to solve real-life problems ?

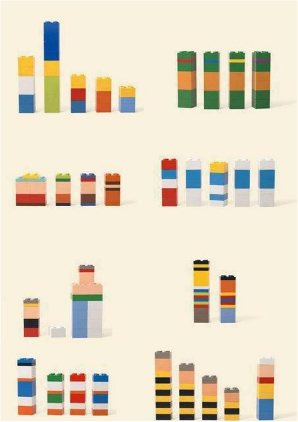
Learning problems are at a cross-section of several applied and science domains :

- Machine learning arose as a subfield of Artificial Intelligence, Computer Science... Emphasis on (large scale) implementations and applications : algorithm centered
- Statistical learning arose as a subfield of Statistics, Applied Maths, Signal Processing, . . . Emphasis on models and their interpretability : model centered

☞ There is much overlap : Data Science

## Learning problem

Often a problem of discovering/learning relevant features, and making "sense" of them...



# Definitions of Learning

### Machine Learning in Computer Science

Tom Mitchell (The Discipline of Machine Learning, 2006)

A computer program CP is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

### Key points

- Experience E : data and algorithms
- Performance measure P : mathematical loss function... or intuitive observation !
- tasks T : utility
  - automatic translation
  - playing Go
  - ... doing what human does

## Experience E : the data !

### Type of data : qualitatives / ordinales / quantitatives variables

| | |
|---|---|
| text | strings |
| speech | time series |
| images/videos | 2/3d dependences |
| networks | graphs |
| games | interaction sequences |
| | ... |

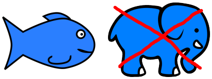
### Big data (volume, velocity, variety, veracity)

Data are available "in the wild" but...

- importance of preprocessings (cleaning up, normalization, coding,...) Data preparation is a science in itself !
- importance of a good representation : from raw data to (usually) vectors

## Objective and performance measures P

- Performance is usually measured by a mathematical loss function
- "Learning" ⇔ minimizing the loss function on some training data
- Loss function design may not be obvious and the choice of function has tremendous effect ! (again, a field in itself)
  - Classifying fish/elephant : OK 😀
  - Playing Go : hmmm. . . 🤔 (Alphago, 2015)
  - "Solving the Minecraft diamond challenge" : ouch ! 😱 (DreamerV3, 2023)



### Generalization

- Perform well (minimize P) on new data (fresh data, i.e. unseen during training)

# Table of Contents

# Classification vs regression

## Variable terminology

**"Most often" :**

- **observed data** called input variables, predictors or features, usually $x$
- **data to predict** called output variables, or responses, usually $y$

Example : $x$ an image of an animal, $y$ a class "cat/dog/other"

## Type of prediction problem : regression vs classification

Depending on the type of the output variables $y$

- quantitative data (continuous, e.g. age, height) : regression
- categorical data (discrete variables, e.g. cat/dog) : classification

Two very close problems. Many hybrid/not-as-easity-categorized tasks !

## Training, testing, generalization

- Training set : labelled data $(x_1, y_1), \ldots, (x_n, y_n)$
- Test data : new $x$ unseen before, predict $y$

*Observed data*

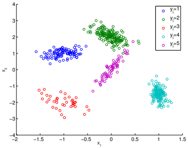
*?*

- "Learning" : deriving a function of prediction / rule of classification $\hat{f} : \mathcal{X} \to \mathcal{Y}$ using only the training data
- Generalization : obtaining a good performance on test data

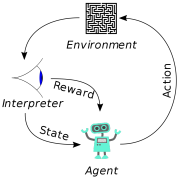
# Three main categories of ML

- Supervised learning : $\mathcal{T} = ((x_1, y_1), \dots, (x_n, y_n))$ input/output couples are available to learn.
  - Goal : **Predict $y$ on a new $x$.**
  - Ex : labelling images.

- Unsupervised learning : $\mathcal{T} = (X_1, \dots, X_n)$ only the inputs are available.
  - Goal : **discovering "interesting structures" in the data** (often used later in other tasks).
  - Ex : grouping users by preferences in a social network



- Reinforcement learning : the algorithm take sequential actions (streaming), receive feedback from the environment.
  - Goal : **take relevant actions, maximize "reward"**.
  - Ex : play Go



## Many variants/overlap

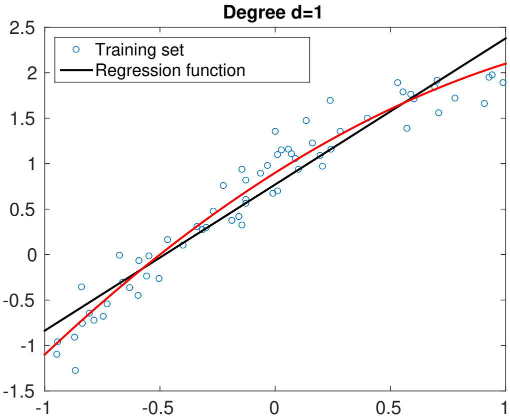
- Semi-supervised : some training data are labelled, some are unlabelled. Exploit both during training. $\rightarrow$ **Labelled data is expensive, unlabelled is cheap !**
- Transfer learning : use features from other tasks (pre-trained models).
- Self-supervised : unsupervised methods are used to produce "labels" used in downstream supervised training (often, go back and forth).
- etc, etc

# Table of Contents

# Table of Contents

# Linear regression

Fit an affine line to the data :



## Linear regression

The prediction rule is :

$$\hat{f}(x) = f_{\beta,b}(x) = \beta^\top x + b$$

where $\beta \in \mathbb{R}^d$, $b \in \mathbb{R}$ are unknown parameters.

### Learning problem $\Leftrightarrow$ Estimation of $\beta, b$

Done by *minimizing* the "empirical risk" on the training data

$$\hat{\mathcal{R}}(\beta, b) = n^{-1} \sum_{i=1}^{n} L(f_{\beta,b}(x_i), y_i)$$

for some loss function $L$.

- Least-Square :

  $$L(\hat{y}, y) = (\hat{y} - y)^2$$

  One of the most classical choice for regression.

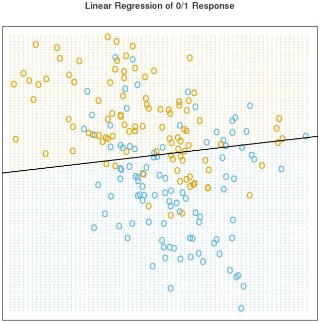
  - NB : in this case, $\beta, b$ have an explicit expression. It is generally not the case, and they must be found with optimization algorithms like gradient descent.

# What about (binary) classification ?

- Binary output variables : $y \in \{-1, +1\}$,
- the previous linear rule can be adapted with a simple threshold

$$\hat{y} = \begin{cases} +1 & \text{if } \hat{f}(x) \geq 0 \\ -1 & \text{if } \hat{f}(x) < 0 \end{cases}$$

- Black solid line is the decision boundary $\hat{f}(x) = \beta^\top x + b = 0$.




*Example of a binary classification problem in* $\mathbb{R}^2$.

# Parametric classification : simple linear model

We still use the empirical risk approach

$$\hat{\mathcal{R}}(\beta, b) = n^{-1} \sum_{i=1}^{n} L(f_{\beta,b}(x_i), y_i)$$

- Least-Square : $L(f(x), y) = (f(x) - y)^2$ makes more "sense" for regression, but still give reasonable results.
- Logistic-Regression : transform $f(x)$ into a probability that $y = 1$

$$p(x) = \frac{1}{1 + e^{-f(x)}} \in [0, 1]$$

such that $p(x) \to 1$ when $f(x) \to \infty$, and $p(x) \to 0$ when $f(x) \to -\infty$.Then, use the negative log-likelihood of a Bernoulli variable that takes value 1 with probability $p(x)$ (see later) :

$$L(f(x), y) = \begin{cases} -\log p(x) & \text{if } y = 1, \\ -\log(1 - p(x)) & \text{if } y = -1 \end{cases}$$

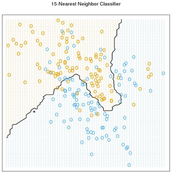
or, more succintly, $L(f(x), y) = \log(1 + e^{-yf(x)})$

## Table of Contents

# Parametric vs. non-parametric

- Linear prediction is parametric : there is a fixed number of parameters to learn, that does no depend on the number of samples in the training set.
  - Linear models
  - Discriminant analysis
  - Neural networks, etc.


- Some algorithms are non-parametric : the number of "parameters" to learn depend on the number of samples in the training set
  - k-NN (just after)
  - Decision tree (not in this course)
  - Kernel methods



# $k$ Nearest Neighbors ($k$-NN) for classification

- algorithm which is conceptually among the simplest of all machine learning algorithms
- The prediction $\hat{f}(x)$ is simply the majority class of the $k$ closest element in the training set in the training set.
- For the previous *binary* classification task $y \in \{-1, 1\}$ :

$$\hat{f}(x) = \begin{cases} 1 & \text{if } \sum_{i \in N_k(x)} y_i \geq 0 \\ -1 & \text{else} \end{cases}$$

where $N_k(x)$ contains the indices of the $k$ inputs $x_i$ closest to $x$ in the training set.

☞ the number of neighbors $k$ is an *hyperparameter* of the algorithm : its value must be fixed (positive integer) prior to applying the algorithm.

## $k$-NN : remarks

- "Closest" depend on a notion of distance (metric) between points $d(x, x')$. Euclidean distance $\|x - x'\|$ is a classic choice, but many other exist depending on the type of data (tabular, text, images, etc.)
- $k$-NN lose in efficacity in high dimension ("curse of dimensionality"). Intuitively, high-dimensional spaces are mostly empty, and all points are "far" from each other. Distances and predictions are not very relevant anymore.
  - ☞ Dimension-reduction like PCA can (sometimes) be a solution...
- $k$-NN has no "training", all the work is done at prediction. It is a lazy algorithm.

| | Training | Prediction ($n'$ points) |
|---|---|---|
| Linear Reg. (naive) | $O(nd^2 + d^3)$ | $O(n'd)$ |
| $k$-NN (naive) | N/A | $O(nn'd \log(k))$ |

- $k$-NN can be (very) costly for big datasets, there are *numerous* ways of accelerating it : trees to compute approximate distance (not lazy anymore !), dimension reduction, sketching...

# Nadaraya-Watson estimator

- Idea : Instead of taking the $k$-closest samples, take all samples, but with a weighted average, where the weights are lower for points that are far :

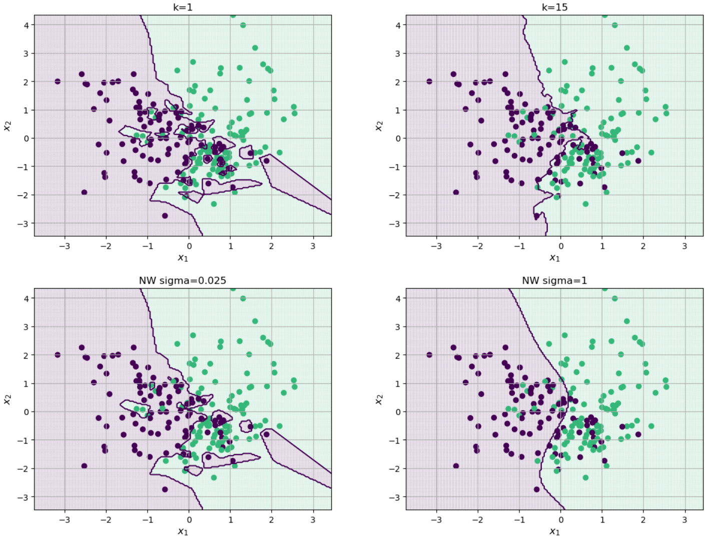
$$f_{NW}(x) = \sum_{i=1}^{n} w_i(x) x_i \text{ with } \sum_i w_i(x) = 1$$

for classification : $\hat{f}(x) = 1$ if $f_{NW}(x) \geq 0$.

- $w_i(x)$ are generally defined through a kernel function $k_\sigma(x, x_i)$ that decreases with the distance between points, which is then normalized. Eg Gaussian

$$k_\sigma(x, x_i) = e^{-\frac{\|x - x_i\|^2}{2\sigma^2}}, \quad w_i(x) = \frac{k_\sigma(x, x_i)}{\sum_{j=1}^{n} k_\sigma(x, x_j)}$$

# K Nearest-Neighbors vs NW



# Table of Contents

## Empirical Risk Minimization (ERM) : a systematic approach

- Parametric model : seek a function $f_\theta(x)$ parametrized by $\theta \in \Theta$ such that $y \approx f_\theta(x)$ (for regression), or $yf_\theta(x) \gg 0$ (for classification)
  - Linear models, neural networks, etc.
- Generally done through a loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$. The ultimate goal is to minimize the expected risk (expected error, generalization error...) :

$$\mathcal{R}(\theta) = \mathbb{E}_{(x,y)} L(f_\theta(x), y)$$

  where the expectation is taken over the joint distribution of $(x, y)$. This ensure good generalization (theoretically, the best possible).
- Since the true expectation is not known in practice, we use the empirical risk over the training data (training error) :

$$\hat{\mathcal{R}}(\theta) = \frac{1}{n} \sum_i L(f_\theta(x_i), y_i)$$

  - if $(x_i, y_i)$ are independently and identically distributed (i.i.d.), the expected risk and empirical risk are "close" (law of large numbers)

## Loss functions

- $L(f(x), y)$ is often of the form $L(f(x) - y)$ for regression and $L(yf(x))$ for (binary) classification

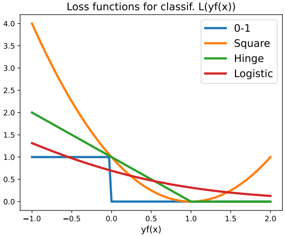
### Regression

The square loss $L(f(x), y) = (f(x) - y)^2$ is by far the most popular approach

- the expected risk is then also known as the Mean Square Error (MSE)

### Binary classification

- 0-1 loss : $1_{yf(x)>0}$. "Ideal" in some sense, but combinatorial and NP-hard to optimize.
- Square loss : $(1 - yf(x))^2$. Easy to optimize, but not very natural for classification.
- Logistic loss : $\log(1 + e^{-yf(x)})$. Very popular (in particular for multiclass classif.), also called "Cross-entropy"
- etc. https://en.wikipedia.org/wiki/Loss_functions_for_classification



## Exercise : True error rate

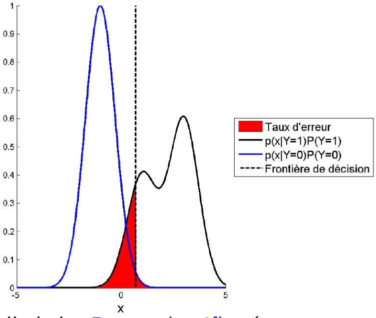
Consider a binary classification problem $y \in \{0, 1\}$ with scalar data $x \in \mathbb{R}$. The $0-1$ loss gives the misclassification probability :

$$\mathcal{R} = \mathbb{E}_{x,y}(1_{f(x) \neq y}) = \mathbb{P}(f(x) \neq y)$$

For a prediction rule $f(x) = 0$ if $x \leq t$, 1 otherwise, where $t$ is a given threshold, and based on the

- class weights $\Pr(y = 0)$, $\Pr(y = 1)$
- class pdf's $p(x|y = 0)$, $p(x|y = 1)$

how can we compute the misclassification probability ?




The optimal classifier for the $0-1$ loss is called the Bayes classifier (see next session).

# Table of Contents

# Optimization

How to find the minimum $\min_\theta \hat{\mathcal{R}}(\theta)$ ?

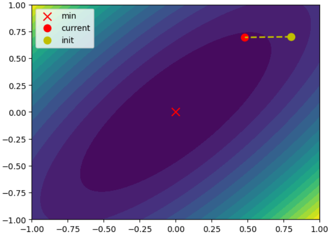
- This is done through optimization algorithms.
  - Optimization is at the core of modern ML !
  - The computational load ("GPU war") is vastly due to training through optimization.
- Overwhelmingly, the dominant approach is gradient descent (and its billions of variants). It is an iterative algorithm that start with some $\theta^{(0)}$ and take a step at each iteration in the direction of steepest descent :

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta \hat{\mathcal{R}}(\theta^{(t)})$$

# Gradient descent

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta \hat{\mathcal{R}}(\theta^{(t)})$$

Quadratic example : $f : x \mapsto x^\top \Sigma x$



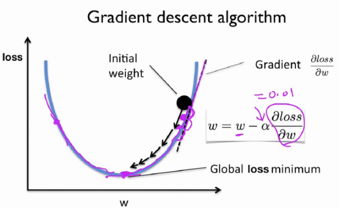
## Remarks

- Modern AI is largely due to the fact that nowadays (Python) libraries can compute gradient automatically through "auto-differentiation". This can be done extremely efficiently with GPUs.

```python
import torch
theta = torch.tensor(something, requires_grad=True)
loss = f(theta) # some function f using only basic operations
loss.backward() # compute the gradient by backprop
theta.grad # contains the gradient

```

- Most of the theory applies to convex optimization. Non-convex optimization is "not supposed to work", but it "does" for modern neural networks. It is still very mysterious !

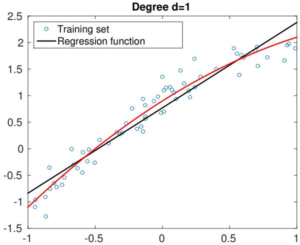


# Table of Contents

# Table of Contents

# Simple linear model classification

A major problem in ML is choosing the function class/hyperparameters of the model.
For instance, consider the linear regression problem




- Is there a room for improvement of the classification performance ?

# Feature engineering

One simple idea is to compute more features, and apply again a linear model in higher dimension.

### Linear regression : model order $p$

E.g. $q$th degree polynomial regression : $p = q + 1$ parameters $\beta_k$ s.t.

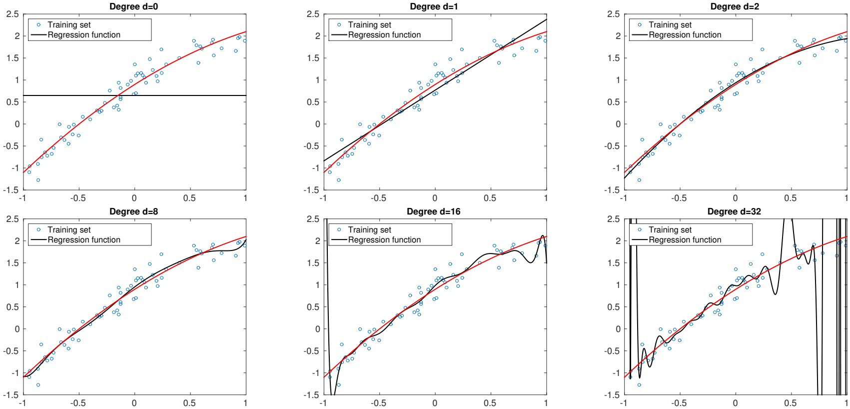
$$\hat{f}(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_q x^q = x_q^\top \beta$$

where

$$x_q = \left[1, x, x^2, \ldots, x^q\right],$$
$$\beta = [\beta_0, \beta_1, \beta_2, \ldots, \beta_q]$$

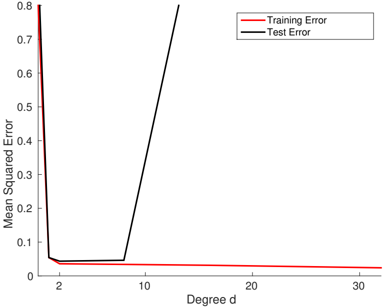
# Linear regression : complexity vs stability

How to choose $q$ ?



# Linear Regression : Test error vs Train Error

When $q$ increases, the training error descreases, but the test error increases. This is the over-fitting phenomenon : the model fits “too much” the training data, and generalizes poorly.

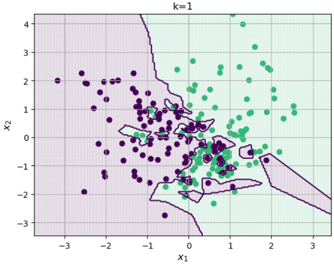



- True error rate (i.e. error rate for test data not used for learning) minimized when $q = 2$ ...
- ... true generative model : order $q = 2$ polynomial (+ white noise)

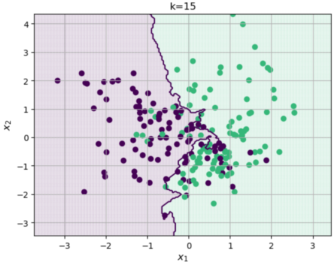
☞ One cannot use the training error to select the model !

# Model Selection : under/over-fitting tradeoff

Is it also true for non-parametric models like $k$-NN ?
Recall the pictures



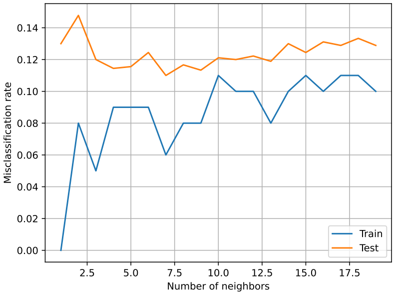

$k = 1$




$k = 15$

- $k = 1 \rightarrow$ training error is always 0 ! Explain why.
- Question : in your opinion, which one gives the best classification rule ?

# $k$-NN : train vs test



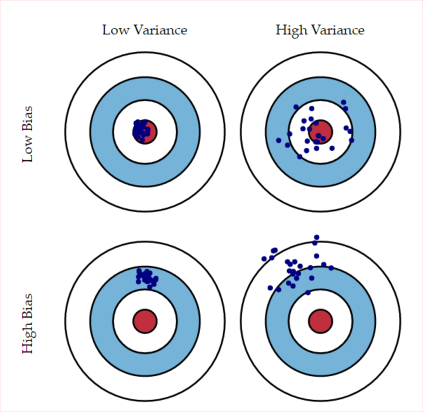

Train and test error (misclassification rate) vs the flexibility of the k-NN algorithm

- when $k = 1$, the model becomes too flexible $\leftarrow$ over-fitting
- when $k$ is large, the model becomes too simple $\leftarrow$ under-fitting
- $k = 7$ is the optimal choice for this dataset

# Fundamental trade-off : Bias vs Variance

## Definition

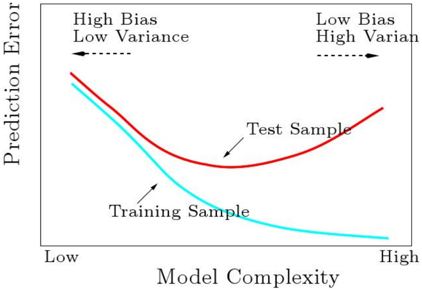
- Bias : how much the model is centered on the true answer
- Variance : how much the model has high stochastic variations



# Fundamental trade-off : Bias vs Variance

## Relation with over/under-fitting

- too simple model (high bias) → under-fitting
  - the model is too simple to fit the true function
- too complex model (high variance) → over-fitting
  - the model overfits the training data, and gives random garbage on test data



## Bias/variance : mathematical definition

- Consider a model
$$y = f^\star(x) + \epsilon,$$
where $\epsilon$ is some centered noise $\mathbb{E}(\epsilon) = 0$.
- Consider a prediction rule $\hat{f}_D(x)$ that depends on some training dataset $D = \{(x_i, y_i)\}_{i=1}^n$.
- Property : we have the decomposition of the MSE
$$\mathbb{E}_{D,\epsilon}\left[\left(y - \hat{f}_D(x)\right)^2\right] = \mathrm{Var}_D\left[\hat{f}_D(x)\right] + \mathrm{Bias}_D\left[\hat{f}_D(x)\right]^2 + \mathrm{Var}\left[\epsilon\right]$$
Proof : see `https://en.wikipedia.org/wiki/Bias-variance_tradeoff`
- Where :
  - $\mathrm{Bias}_D\left[\hat{f}_D(x)\right] = \mathbb{E}_D(\hat{f}_D(x)) - f^\star(x)$ reflects how much the model is centered on $f^\star$
  - $\mathrm{Var}_D\left[\hat{f}_D(x)\right] = \mathbb{E}_D\left[\left(\hat{f}_D(x) - \mathbb{E}_D(\hat{f}_D(x))\right)^2\right]$ reflects how much the model varies with $D$
  - $\mathrm{Var}\left[\epsilon\right]$ is the irreducible part, the level of noise.

## Table of Contents

# Model Assessment/Validation

- Pbm : we have only access to the training data, but we cannot select the model/hyperparameters through the empirical risk alone (overfitting)
- Idea : separate the training set, use part of it to approximate the generalization error (do not include that part in the training !) and use that to select the model.
- This held-out part of the training set is called the validation set.

$\longleftarrow$ Full Dataset $\longrightarrow$

| Train | Validation |
|---|---|

- But
  - only part of the data is used to train the model : to preserve performance, the validation set must not be too large
  - To have a good approximation of the test error, the validation set must be large !
- standard solution : cross-validation, e.g. K-fold Cross-Validation

# K-fold Cross-Validation (CV) : Principle

- Idea : take several different validation sets, for each train a model, average the validation errors.

- In practice split the data in $K$-folds, here $K = 5$ :

| | | | | | |
|---|---|---|---|---|---|
| $\hat{\mathcal{R}}_1(\hat{f}_1, \lambda)$ | Validation | Train | Train | Train | Train |
| $\hat{\mathcal{R}}_2(\hat{f}_2, \lambda)$ | Train | Validation | Train | Train | Train |
| $\hat{\mathcal{R}}_3(\hat{f}_3, \lambda)$ | Train | Train | Validation | Train | Train |
| $\hat{\mathcal{R}}_4(\hat{f}_4, \lambda)$ | Train | Train | Train | Validation | Train |
| $\hat{\mathcal{R}}_5(\hat{f}_5, \lambda)$ | Train | Train | Train | Train | Validation |

  where $\lambda$ is some hyperparameter of the model/method

- Estimate of Test error :

$$\mathrm{CV}(\hat{f}, \lambda) = \frac{1}{K} \sum_{k=1}^{K} \hat{\mathcal{R}}_k(\hat{f}_k, \lambda)$$

- Usually K=5 or 10 is a good trade-off (K=n is called leave-one-out)

# Conclusions

Model assesment methods are an essential tool for data analysis, especially for big datasets involving many predictors

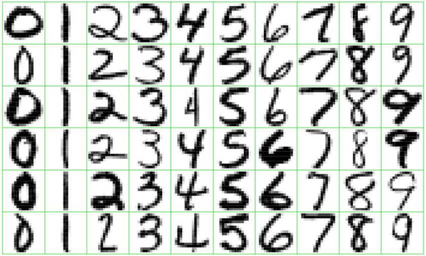
## Cross-validation

- generic and simple procedure that can be used for any supervised problem
- provides a direct estimate of the test error.
- Can be used for model selection and/or hyperparameter tuning
- $K$-fold (with $K = 5$, or $K = 10$) is a standard choice, but there exists many variants depending on the problem, e.g.
  - *stratified* $K$-fold to ensure that all the folds have roughly the same average response value $\leftarrow$ useful for classification to be sure that each fold contains roughly the same proportions of class labels.
  - *hold-out* cross-validation for time-series where a subset (split temporally) of the data is reserved for validating the model performance

# Table of Contents

# Examples of Tasks

## Recognition of handwritten digits (US postal envelopes)



- ☞ Predict the class (0,...,9) of each sample from an image of $16 \times 16$ pixels, with a pixel intensity coded from 0 to 255
- Low error rate to avoid wrong allocations of mails !
- Historically one of the first, most popular image dataset : MNIST

Supervised classification

# Examples of Tasks

## Spams Recognition

### Spam

```
WINNING NOTIFICATION
We are pleased to inform you of the result
of the Lottery Winners International
programs held on the 30th january 2005.
[...] You have been approved for a lump sum
pay out of 175,000.00 euros.
CONGRATULATIONS!!!
```

### No Spam

```
Dear George,
Could you please send me the report #1248 on
the project advancement?
Thanks in advance.

Regards,
Cathia
```

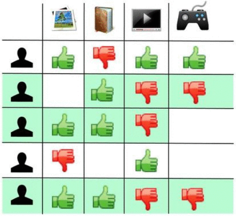
☞ Define a model to predict whether an email is spam or not

- Low error rate to avoid deleting useful messages, or filling the mailbox with useless emails
- One of the most deployed application today ! !

supervised classification

# Examples of Tasks

## Recommender systems



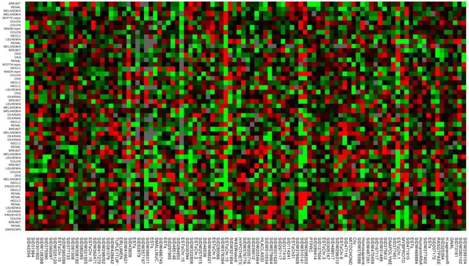
☞ Make personalized user recommendation of : movies, music, purchase, advertisements, social network friends...

- User-based (collaborative filtering), content-based, often hybrid
- (mis)Used all over the Internet, by every large website
- Historically, also important : the Netflix challenge

unsupervised classification/regression

# Examples of Tasks

## DNA-microarrays

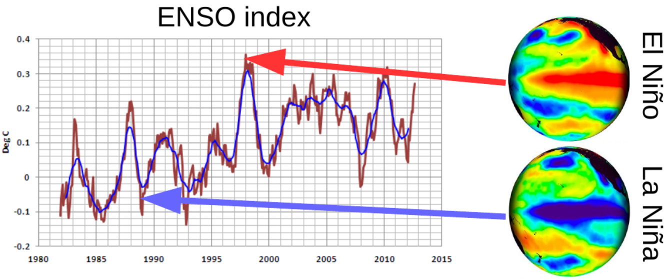


- Genes expression dataset for several thousand individual genes (columns) and tens of samples (rows) $d \gg n$ (huge challenge ! !)
- ☞ Clustering of genes (resp. samples) with similar expression profiles across samples (resp. genes)

unsupervised classification

# Examples of Tasks in Geosciences

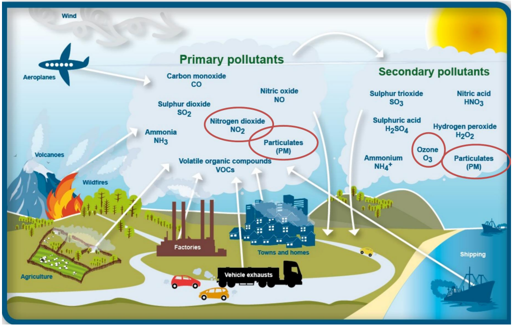
## Prediction of El Niño southern oscillation




- Predict, 6 months in advance, the intensity of an El Niño Southern Oscillation (ENSO) event from ocean-atmosphere datasets (sea level pressure, surface wind components, sea surface temperature, surface air temperature, cloudiness...)

supervised regression (on time series)

## Prediction of pollutant concentrations




☞ Predict pollutant concentrations ($0_3$,$N0_2$,PM10,PM2.5) at time $D_0$+1,+2,+3 from hourly measures timeseries + weather data + chemistry based forecasting models

supervised regression (pollutant concentration prediction) / classification (pollution alert or not)