EE-613: Machine Learning for Engineers

François Fleuret

Sep 16, 2015

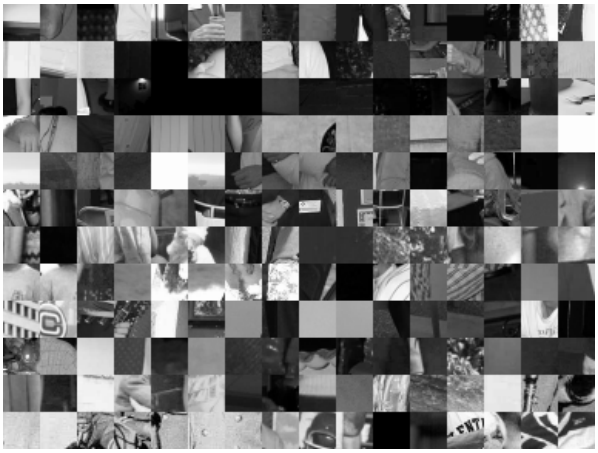
Lecturers

- Sylvain Calinon
  `http://http://www.calinon.ch/`
  sylvain.calinon@idiap.ch

- François Fleuret
  `http://www.idiap.ch/~fleuret/`
  francois.fleuret@idiap.ch

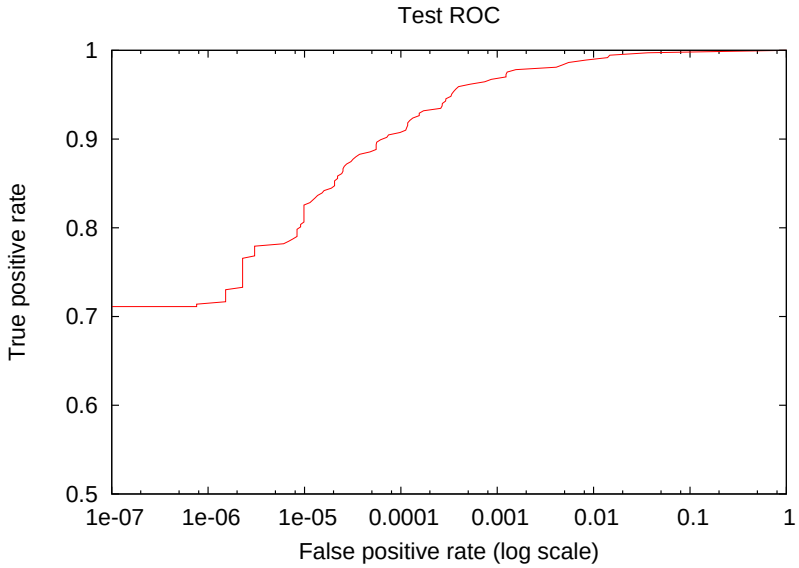
- Jean-Marc Odobez
  `http://www.idiap.ch/~odobez/`
  jean-marc.odobez@idiap.ch

Introduction
What is machine learning

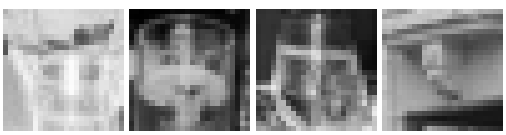


Given a 24×24 gray-scale image, can we predict if it is a face?

Introduction
What is machine learning (cont.)



TP	FP
99%	$\simeq 0.8\%$
80%	$\simeq 10^{-3}\%$



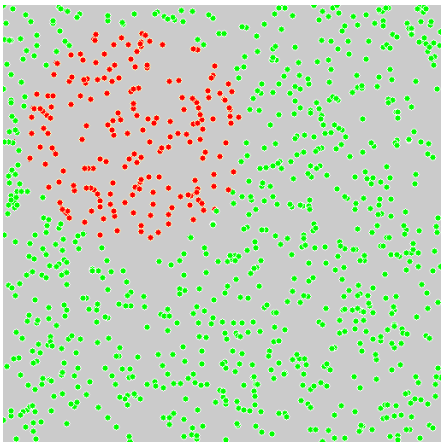
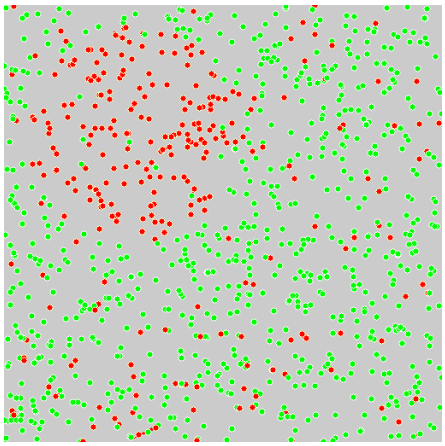
Machine learning aims at designing algorithms to infer the world regularities from a finite set of examples.

In practice, given a set of *training examples* $\mathcal{D}$, build automatically a predictor f^* of a hidden value given the visible signal.

Performance should be good on *test data* which are not available to chose the predictor.

Many real-world applications fit in this framework

Application	Accessible signal	Value of interest
Character recognition	image	text
Scene understanding	image	objects
Speech recognition	sounds	words
Genetic diagnostic	gene expressions	diseases
Biometry	picture/fingerprint	identity
Automatic navigation	radar echoes	obstacles
Surveillance	video streams	activities

(this slide contains videos showing the learning in progress)

Course organization
Plan

Total of 28h course and 28h practical sessions.

- Introduction (2h)
  - What is machine learning about
  - Brief recall on probabilities and gradient descent
  - Performance evaluation
- Generative models (12h)
  - Directed / non-directed models
  - Conditional independence, naive Bayesian
  - Belief propagation
  - k-Mean + GMM + E-M
  - PCA + probabilistic PCA
  - HMM + extensions
  - Sub-space clustering

Course organization
Plan

- Regression (4h)
  - Least-square + weighted least-square
  - GMR + GPR
- Discriminative models (6h)
  - k-NN
  - SVMs and Kernelization (perceptron, PCA, etc.)
  - Perceptron, MLP, and Convolution networks
  - Decision trees
- Meta-algorithms (4h)
  - Bagging and Boosting
  - Feature selection, Regularization and sparsity

Course organization
Evaluation

Evaluation will consists of a 2h exam on the course content.
Some of the questions will be about the practical sessions.

You are more than welcome to send by mail results about the practical sessions so that we can provide a feed-back. However they will not be individually graded.

Crash course in probabilities

Crash course in probability
Measured space

We recall here a few definitions and properties of the probability theory for discrete sets first. Probabilities on $\mathbb{R}^d$ involve a bit more technicalities.

Given a discrete set Ω, a probability measure on it is a function

$$P : \mathcal{P}(\Omega) \to \mathbb{R}_+$$

such that

- $P(\emptyset) = 0$ and $P(\Omega) = 1$
- $\forall (A_n)_{n\in\mathbb{N}} \in \mathcal{P}(\Omega)^{\mathbb{N}}, \forall i \neq j, A_i \cap A_j = \emptyset, P(\cup_n A_n) = \sum_n P(A_n)$

We have to define a family of subsets of Ω called the **events**. We consider in what follows all the possible subsets as events.

We can define the conditional probability of an event given another event as

$$\forall A, B \subset \Omega,\ P(A \mid B) = \frac{P(A \cap B)}{P(B)}.$$

It is related to prediction if B is an observed event ("inherits one mutant p53 allele"), and A an event of interest ("will develop pancreatic cancer").

Given a set S, a S-valued random variable is a mapping:

$$X : \Omega \to S$$

and we define

$$P(X \in A) = P(X^{-1}(A)).$$

The family of all these probabilities is the **distribution** of X.

In the discrete case, it is completely defined by the point-wise probabilities

$$(P(X = x))_{x \in S}$$

Different random variables can have the same probability distribution. For instance, with

$$\Omega = \{1, 2, 3, 4\}$$

and

$$P(\{1\}) = P(\{2\}) = P(\{3\}) = P(\{4\}) = 0.25,$$

the following three random variables X, Y, and Z all have the same probability distribution, uniform on $\{0, 1\}$.

	1	2	3	4
X	1	1	0	0
Y	1	0	1	0
Z	1	0	0	1

Given a discrete random variable X, we define its expectation

$$E(X) = \sum_x P(X = x)\, x,$$

and similarly we define the conditional expectation

$$E(X \mid A) = \sum_x P(X = x \mid A)\, x.$$

The same notation is used for the expectation of the function of a random variable:

$$E(f(X)) = \sum_x P(X = x)\, f(x).$$

Two random variables X and Y are **independent** if knowing the value of one does not say anything about the other.

This can be formulated as

$$\forall A, B \subset S,\ P(X \in A, Y \in B) = P(X \in A)P(Y \in B).$$

We call **joint** distribution the distribution of (X, Y) and **marginal** distributions the distributions of X and Y individually.

The marginals alone do not characterize the joint distribution.

	$X=0$	$X=1$
$Y=0$	0.25	0.25
$Y=1$	0.25	0.25

	$X=0$	$X=1$
$Y=0$	0.5	0.0
$Y=1$	0.0	0.5

Independence is critical for learning and representation.

The joint distribution of a family of S-valued random variables $X_1, \ldots, X_N$ is completely determined by $N(|S| - 1)$ probabilities if they are independent, and $|S|^N - 1$ otherwise.

Making assumptions of independence or conditional independence is often key in modeling real-world data.

Given two discrete and independent random variables X and Y, we have

$$\begin{aligned} E(XY) &= \sum_{x,y} P(X = x, Y = y)\, x\, y \\ &= \sum_{x,y} P(X = x) P(Y = y)\, x\, y \\ &= \sum_{x} P(X = x)\, x \sum_{y} P(Y = y)\, y \\ &= E(X)E(Y) \end{aligned}$$

The computation is strongly reduced by the assumption of independence.

In many practical situations, it is easier to model the observation given the hidden value $P(X = x \mid Y = y)$ than the contrary.

For instance X r.v. on $[0, 1]^3$ the color of a pixel, and Y r.v. on $\{0, 1\}$ the presence of skin at that location of the image.

In such a case, we can use Bayes' law to obtain the quantity of interest

$$P(Y = y \mid X = x) = \frac{P(X = x \mid Y = y)}{P(X = x)} P(Y = y)$$

If Y is finite, we can normalize numerically and we do not need $P(X = x)$.

The formalization through the notion of measured space allows to precisely define the dependence between random variables.

This is for theoretical construction. **In practice, the measured space is often implicit, and one works with the distributions in the value space and independence assumptions.**

Continuous variables involve a bit more problems. In particular, we have to define carefully the events. We will consider only **continuous** probability distributions, to which correspond **probability density functions**:

$$P(A) = \int_A \mu(u)du$$

For instance, if X is a random variable of normal distribution, with mean m and standard deviation s, we have

$$P(X \in [a, b]) = \int_a^b \frac{1}{\sqrt{2\pi}s} e^{-\frac{(x-m)^2}{2s^2}} dx$$

In $\mathbb{R}$, the cumulative distribution function (cdf) is defined as

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} \mu(u)du.$$

An important theorem behind statistical learning is the Central Limit Theorem:

Given $(X_n)_{n\in\mathbb{N}}$ i.i.d, with $E(X_n) = \mu$ and $V(X_n) = \sigma^2 < +\infty$, then

$$\sqrt{n}\left(\frac{1}{n}\sum_{k=1}^{n} X_k - \mu\right) \xrightarrow{d} \mathcal{N}(0, \sigma^2).$$

Note that this is a convergence in distribution.

In machine learning, this result is useful to model the behavior of certain quantities, such as an error estimate.

Other theorems provide exact bounds in probability (e.g. Hœffding's inequality).

Gradient descent

Given a functional

$$f : \mathbb{R}^D \to \mathbb{R}$$

the core idea of gradient descent is to use order 1 information to find a path to a (local) minimum.

Given an $x \in \mathbb{R}^D$, what is a reasonable "better x" ?

With a crude approximation of f

$$f(y) \simeq f(x) + \nabla f(x)^T (y - x) + \frac{1}{2\eta} \|y - x\|^2$$

we get

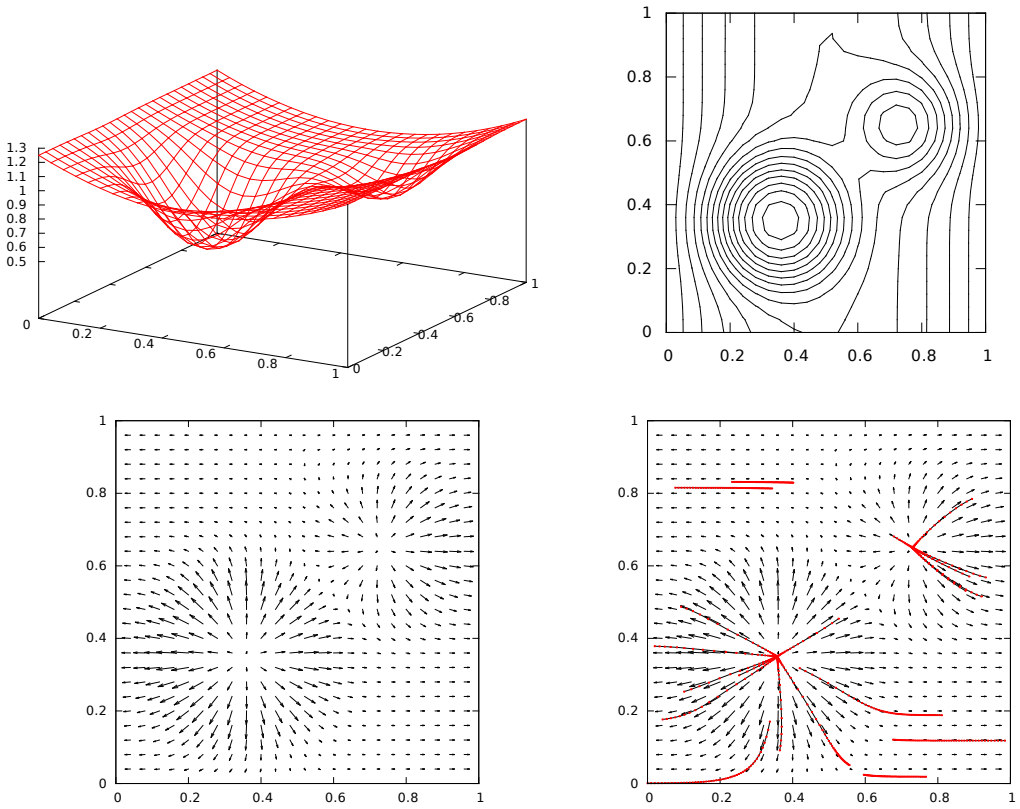
$$\underset{y}{\operatorname{argmin}}\, f(x) = x - \eta \nabla f(x).$$

The resulting iterative rule takes the form of:

$$x_{n+1} = x_n - t_n \nabla f(x_n).$$

This strategy finds a **local** minimum, and choosing x_1 and t_n are critical.

Gradient descent
Example



Gradient descent
Variants

Multiple variations exist around this simple recipe:

- Adaptive η_n: Either fix a decreasing dynamic for η_n, or take into account f variation.
- Line-search: At every step, use

$$\eta_n = \underset{t}{\operatorname{argmin}}\, f(x_n - t\nabla f(x_n))$$

- Conjugate gradient: Do not use ∇f, but a direction updated at every step.
- Natural gradient: Replace the quadratic term in

$$f(y) \simeq f(x) + \nabla f(x)^T (y - x) + \frac{1}{2\eta}\|y - x\|^2$$

by something more fitting to the problem at hand.

In machine learning, the functional to minimize f very often takes the form of a large sum

$$f(x) = \sum_{k=1}^{K} f_k(x)$$

in which case the gradient is

$$\nabla f(x) = \sum_{k=1}^{K} \nabla f_k(x)$$

with a computational cost $O(K)$ at each step.

If the family of f_n is redundant, this is sub-optimal, since we could use a larger step-size with a partial sum of the f_n.

This argument motivates the use of the **stochastic gradient descent**:

$$x_{n+1} = x_n - \eta_n \nabla f_{k_n}(x)$$

which, under reasonable assumptions on the f_k and η_n converges properly.

This strategy allows to deal with extremely large training sets, and is central in all “large-scale” learning techniques.

What is machine learning

Three types of learning
Introduction

Let p be the true distribution of our data, and

$$\mathcal{D} = \{Z_1, \ldots, Z_N\} \in \mathcal{Z}^N,$$

the training examples, that we postulate i.i.d of distribution p.

There are three principal types of predictions:

- Classification
- Regression
- Density estimation

- Classification, where $\mathcal{Z} = \mathbb{R}^D \times \{1, \ldots, C\}$,
  and we want to estimate $\text{argmax}_y P(Y = y \mid X = x)$
  Object recognition, cancer detection, speech processing.
- Regression, where $\mathcal{Z} = \mathbb{R}^D \times \mathbb{R}$,
  and we want to estimate $E(Y \mid X = x)$.
  Customer satisfaction, stock prediction, epidemiology.
- Density estimation, where $\mathcal{Z} = \mathbb{R}^D$,
  and we want to estimate $p(z)$
  Data visualization, pre-processing.

Learning consists of finding a "good" functional in a pre-defined set of functionals $\mathcal{F}$. For example:

- For classification:

$$f(x; w_1, \ldots, w_C) = \underset{y}{\text{argmax}} \langle w_y, x \rangle$$

- For regression:

$$f(x; \alpha_1, \ldots, \alpha_K) = \sum_k \alpha_k h_k(x)$$

- For density estimation:

$$q(x; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp\left(-\frac{(z-\mu)^T \Sigma^{-1} (z-\mu)}{2} \right)$$

Three types of learning
Loss

We define the "good" functionals through a loss function

$$L : \mathcal{F} \times \mathcal{Z} \to \mathbb{R}_+$$

such that $L(f, z)$ indicates how wrong f is on sample z.

For example:

- For classification:

$$L(f, (x, y)) = 1_{\{f(x) \neq y\}}$$

- For regression:

$$L(f, (x, y)) = (f(x) - y)^2$$

- For density estimation:

$$L(q, z) = -\log q(z)$$

Expected and empirical risks
Definitions

We are looking for an f with a small expected risk

$$R(f) = E_{Z \sim p}\left(L(f, Z)\right)$$

which means that our learning procedure should pick

$$f^* = \operatorname*{argmin}_{f \in \mathcal{F}} R(f).$$

Unfortunately this quantity is not available, but we can use the training samples to compute the empirical risk:

$$\hat{R}(f; \mathcal{D}) = \hat{E}_{\mathcal{D}}(L(f, Z)) = \frac{1}{N} \sum_{n=1}^{N} L(f, Z_n)$$

We have

$$\begin{aligned} E_{Z_1,\dots,Z_N\sim p}\left(\hat{R}(f;\mathcal{D})\right) &= E_{Z_1,\dots,Z_N\sim p}\left(\frac{1}{N}\sum_{n=1}^{N} L(f,Z_n)\right) \\ &= \frac{1}{N}\sum_{n=1}^{N} E_{Z_n\sim p}\left(L(f,Z_n)\right) \\ &= \frac{1}{N}\sum_{n=1}^{N} E_{Z\sim p}\left(L(f,Z)\right) \\ &= E_{Z\sim p}\left(L(f,Z)\right) \\ &= R(f) \end{aligned}$$

Hence the empirical risk is a non-biased estimator of the expected risk.

Finally, given $\mathcal{D}$, $\mathcal{F}$, and L, "learning" aims at computing

$$f^* = \operatorname*{argmin}_{f\in\mathcal{F}} \hat{R}(f;\mathcal{D})$$

- Can we bound $R(f)$ with $\hat{R}(f,\mathcal{D})$?

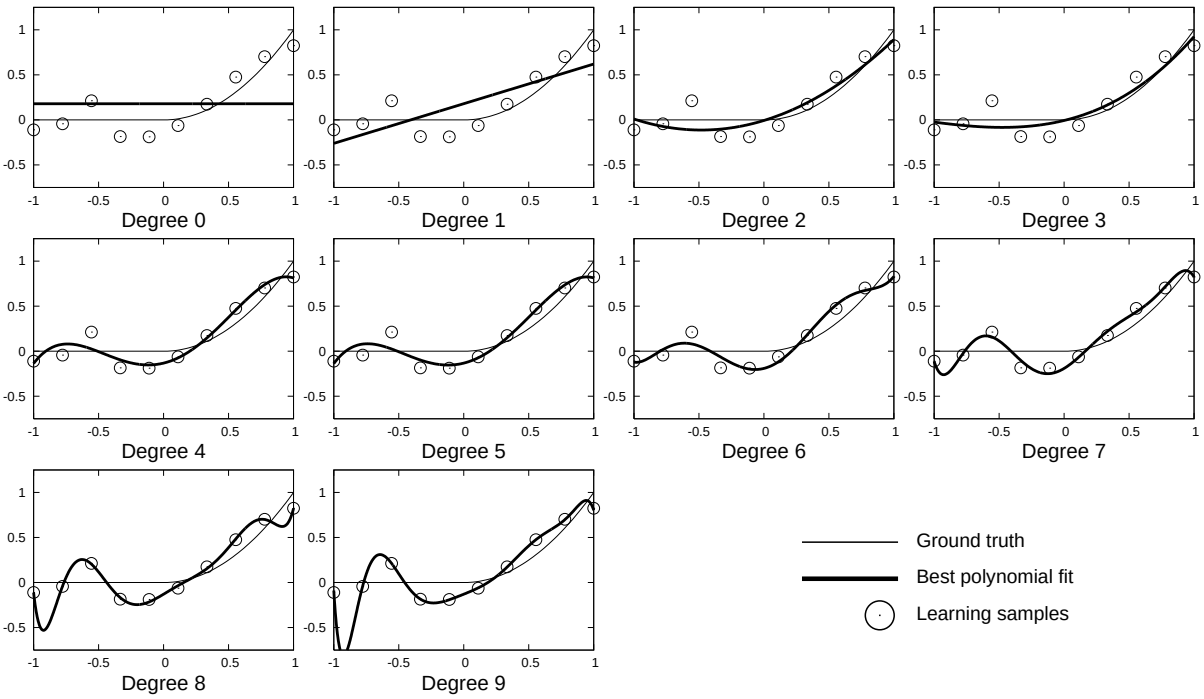
  Yes if f is not chosen using $\mathcal{D}$. Since the Z_n are independent, we just need to take into account the variance of $\hat{R}(f,\mathcal{D})$.

- Can we bound $R(f^*)$ with $\hat{R}(f,\mathcal{D})$?

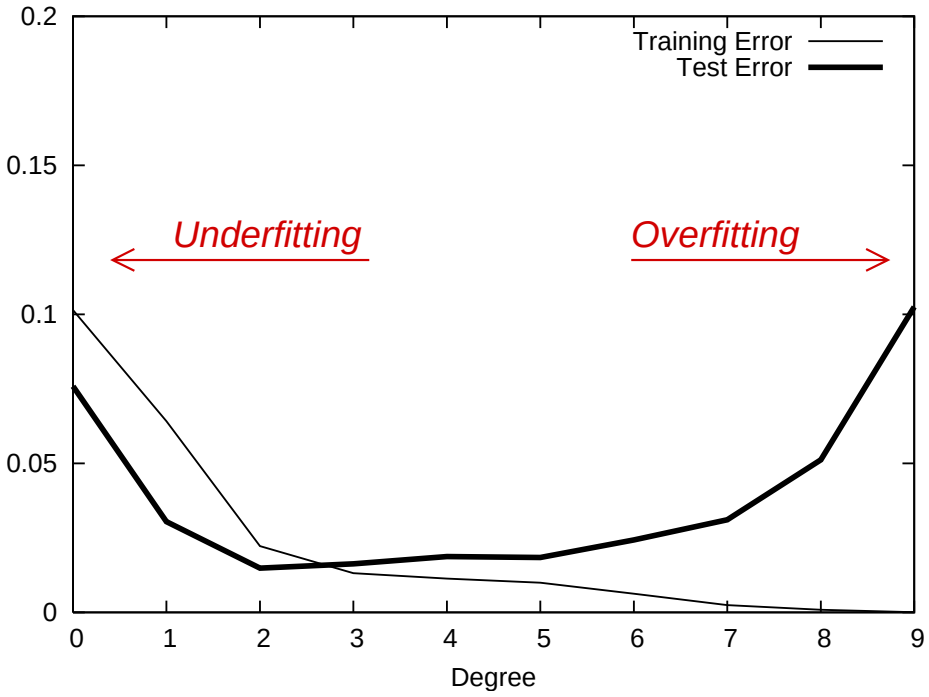
  Unfortunately, not simply, and not without additional information about $\mathcal{F}$. For instance if $|\mathcal{F}| = 1$, we can.

Under and Over-fitting
Example: Polynomial regression

We fix $\mathcal{D}$ and increase the space $\mathcal{F}$.

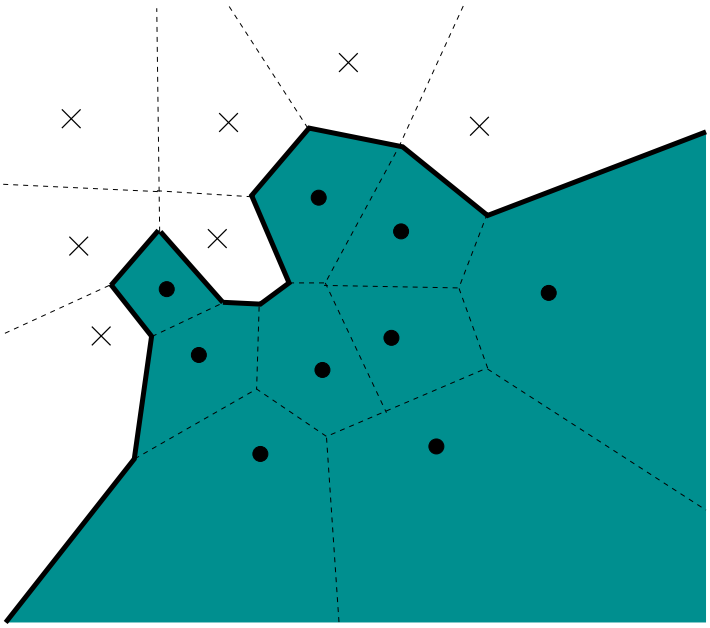


Under and Over-fitting
Example: Polynomial regression (cont.)

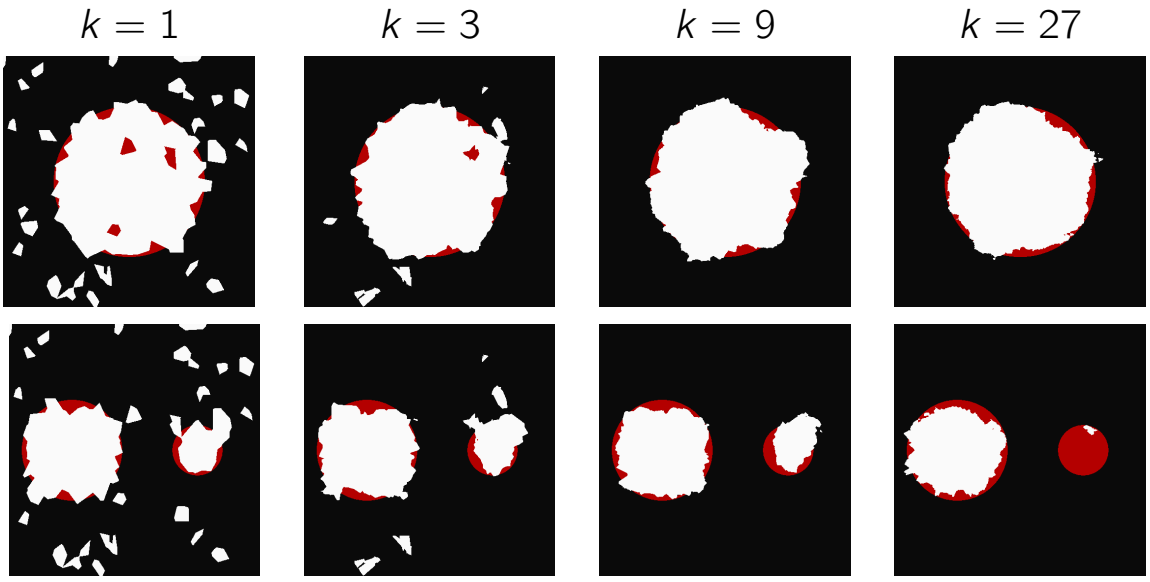


Under and Over-fitting
Example: k-Nearest Neighbors

The *nearest neighbour* classifier predicts that the class of an X is the class of the closest training example.



Under and Over-fitting
Example: k-Nearest Neighbors (cont.)



k-Nearest Neighbors (500 training points, 5% flip-noise).

We observe that when the "richness" of $\mathcal{F}$ increases, the gap between the expected and the empirical risk increases.

To bound $R(f^*)$ from $\hat{R}(f^*, \mathcal{D})$, we need an additional term to reflect the "richness" of $\mathcal{F}$.

In classification, we can consider the capacity of $\mathcal{F}$, which is the maximum size of a set which can be arbitrarily labelled by a function from $\mathcal{F}$.

For regression, we can decompose the expected error as the sum of a bias and a variance term:

$$E_{\mathcal{D}}\left((f^*(x) - f(x))^2\right) = \underbrace{\left(E_{\mathcal{D}}(f^*(x)) - f(x)\right)^2}_{\text{Bias}} + \underbrace{V_{\mathcal{D}}(f^*(x))}_{\text{Variance}}$$

Increasing the capacity reduces the bias, since f^* fits better the data on average, but increases the variance, since f^* varies a lot with the training data.

The main strategies to control over-fitting is through regularization:

- Impoverish the space $\mathcal{F}$ (less functionals, early stopping)
- Make the choice of f^* less dependent on data (penalty on coefficients, margin maximization, ensemble methods)

A machine learning algorithm combines

- A space $\mathcal{F}$
- A regularization term $H(f)$
- An algorithm to compute $\operatorname{argmin}_{f \in \mathcal{F}} \hat{R}(f; \mathcal{D}) + H(f)$

For instance the classical perceptron, and linear SVMs share the same $\mathcal{F}$.

Similarly for GMM and Parzen windows.

Many variants of ANNs differ only through the H term or the optimization algorithm.

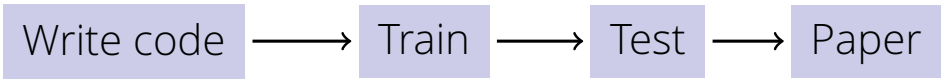
The main practical issue to address is the trade-off between under and over-fitting.

- Under-fitting: No available functional is consistent with the data we have.
- Over-fitting: The chosen functional is extremely good on the training data, but models irrelevant random perturbations.

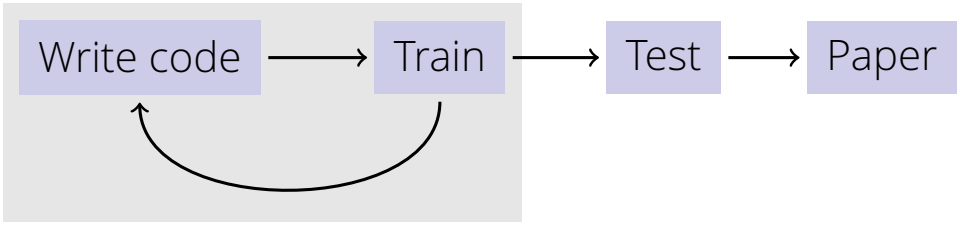
The art of machine learning is to combine expertise to build a sound space of predictors, and good statistical techniques to pick the best one.

Proper evaluation protocols

The ideal development cycle is

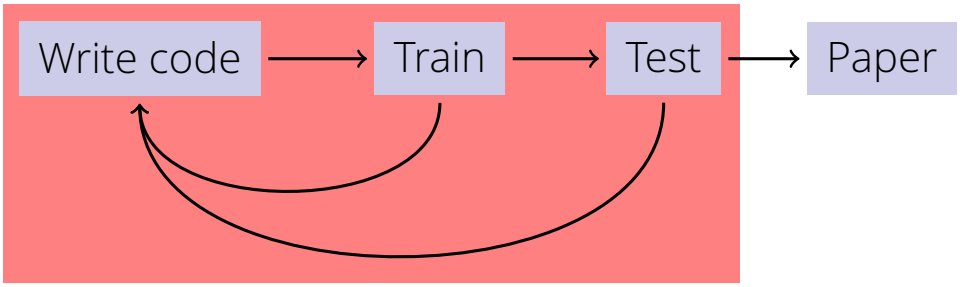


or in practice something like

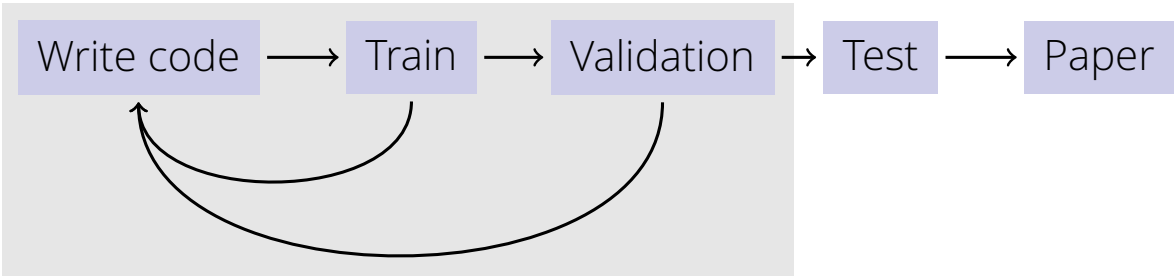


There may be over-fitting, but it does not bias the final performance evaluation.

Unfortunately, it often looks like



This should be avoided at all costs. The standard strategy is to have a separate validation set for the tuning.



When data is scarce, we can use cross-validation. It consists of repeatedly splitting the training data into a train and a validation set, and averaging the risk estimate through the multiple folds.

There does not exist any unbiased and universal estimator of the variance of k-fold cross-validation valid under all distributions (Bengio & Grandvalet 2004).

Other typologies

Discriminative vs. generative
Example: Gender prediction

The discriminative methods produce the value of interest without modeling the data structure.

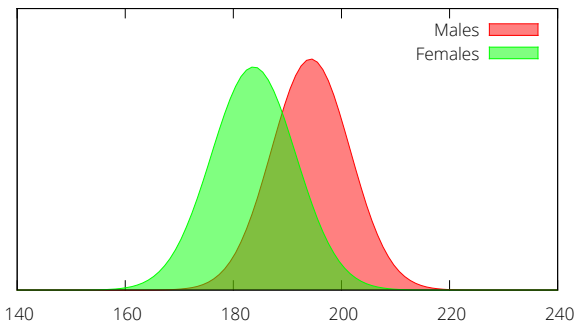
The generative approaches rely on a model of the data, even if it is not the quantity of interest.

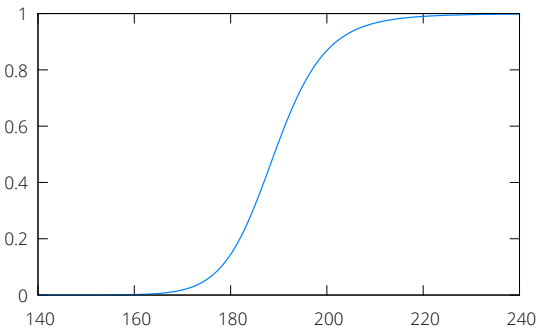
Example: Can we predict a Chinese basketball player's gender from his/her height?

Females		Males	
190	180	190	195
182	193	193	184
188	179	199	190
184	186	200	203
196	185	192	205
173	169	190	201

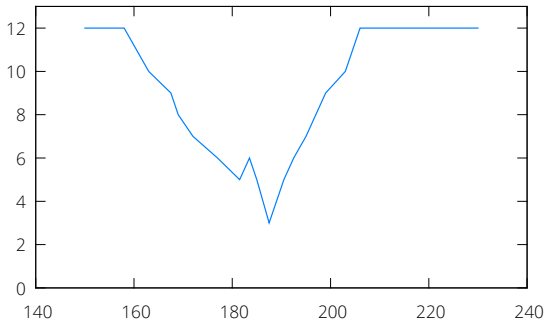
Discriminative vs. generative
Example: Gender prediction (cont.)

We can either model $P(H \mid G)$ and from that derive $P(G \mid H)$ using Bayes' law.





But we can also directly look for the best threshold:



Other typologies
Supervision and parametrization

- Supervised vs. unsupervised

  Supervised learning has access to the values to predict.

- Parametric vs. non-parametric

  Fit a finite (small) number of parameters vs. select a model with a large (possibly infinite) number of degrees of freedom.

- Linear algebra
- Probabilities (modeling, bounds)
- Classical statistics (performance estimates)
- Signal processing (feature design, pre-processing)
- Optimization (estimation of the model's parameters)
- Algorithmic (efficient implementations)
- System programming (large-scale learning)

Practical session

We are interested in today's practical session in regression with a functional basis in C++. Given

$$f_k : \mathbb{R} \to \mathbb{R},\ k = 1, \dots, K,$$

and a training set

$$(x_n, y_n) \in \mathbb{R} \times \mathbb{R},\ n = 1, \dots, N.$$

With the L^2 error

$$L(\alpha_1, \dots, \alpha_K) = \frac{1}{2} \sum_n \left(\sum_k \alpha_k f_k(x_n) - y_n \right)^2,$$

can we find

$$\underset{\alpha_1, \dots, \alpha_K}{\operatorname{argmin}}\ L(\alpha_1, \dots, \alpha_K) \, ?$$

We have

$$\begin{aligned}
\forall r,\ \frac{\partial L}{\partial \alpha_r} &= \frac{1}{2} \sum_n \frac{\partial}{\partial \alpha_r} \left(\sum_k \alpha_k f_k(x_n) - y_n \right)^2 \\
&= \sum_n f_r(x_n) \left(\sum_k \alpha_k f_k(x_n) - y_n \right) \\
&= \sum_n f_r(x_n) \sum_k \alpha_k f_k(x_n) - \sum_n f_r(x_n) y_n \\
&= \sum_k \alpha_k \sum_n f_r(x_n) f_k(x_n) - \sum_n f_r(x_n) y_n
\end{aligned}$$

So, solving

$$\forall r,\ \frac{\partial L}{\partial \alpha_r} = 0$$

amounts to solve

$$\begin{bmatrix} \sum_n f_1(x_n) f_1(x_n) & \dots & \sum_n f_1(x_n) f_K(x_n) \\ \vdots & \ddots & \vdots \\ \sum_n f_K(x_n) f_1(x_n) & \dots & \sum_n f_K(x_n) f_K(x_n) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_K \end{bmatrix} = \begin{bmatrix} \sum_n f_1(x_n) y_n \\ \vdots \\ \sum_n f_K(x_n) y_n \end{bmatrix}$$

hence

$$\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_K \end{bmatrix} = \begin{bmatrix} \sum_n f_1(x_n) f_1(x_n) & \dots & \sum_n f_1(x_n) f_K(x_n) \\ \vdots & \ddots & \vdots \\ \sum_n f_K(x_n) f_1(x_n) & \dots & \sum_n f_K(x_n) f_K(x_n) \end{bmatrix}^{-1} \begin{bmatrix} \sum_n f_1(x_n) y_n \\ \vdots \\ \sum_n f_K(x_n) y_n \end{bmatrix}$$

The practical session will take place **at 14:15-16:00 in room INN218**. To download the materials and run a test example:

```
wget http://www.idiap.ch/~fleuret/files/EE613/EE613-pw1.tgz
tar zxvf EE613-pw1.tgz
cd ./EE613/pw1
./do.sh -v example
```

The archive contains a file `EE613-pw1.pdf` with the questions.