EE-613: Machine Learning for Engineers

Jean-Marc Odobez

Sep 19, 2019

RESEARCH INSTITUTE

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

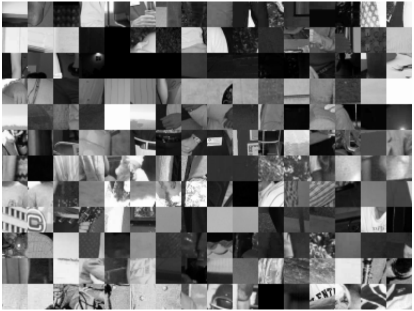
Credit for introductory course: François Fleuret - C. Bishop's Book

Lecturers

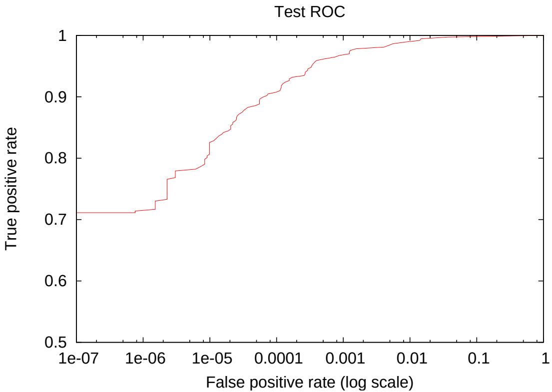
- Sylvain Calinon
  http://http://www.calinon.ch/
  sylvain.calinon@idiap.ch

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

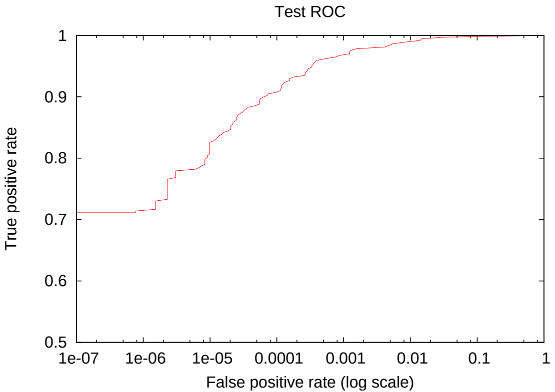
- Jean-Marc Odobez
  http://www.idiap.ch/~odobez/
  jean-marc.odobez@idiap.ch



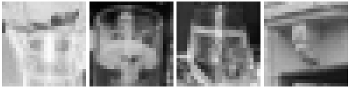
Given a 24×24 gray-scale image, can we predict if it is a face?



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



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.

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.

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

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
  - k-Mean + GMM + E-M
  - HMM + extensions
  - PCA + probabilistic PCA
  - Belief propagation
  - Sub-space clustering

- 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

Labs will be mainly in the form of written exercices or jupyter notebooks

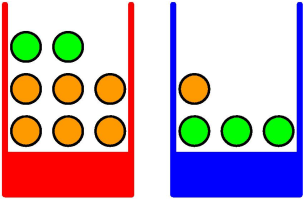
Evaluation will consists of a 2h exam on the course content. **Some of the questions will be about the practical sessions.**

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.

Probabilities

We recall here informaly a few definitions and properties of the probability theory for discrete sets first.

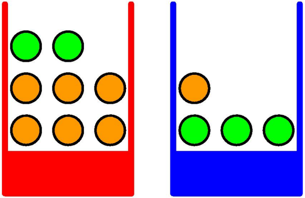
A random variable is a variable that can take several values (events) according to some probability distribution.



Example: having balls within two possible boxes and of two colors, one random variable denoted B represents the identity of the box, and can take two values (r or b). The ball color is another random variable, F, and can take the values g or o.

We recall here informaly a few definitions and properties of the probability theory for discrete sets first.

A random variable is a variable that can take several values (events) according to some probability distribution.

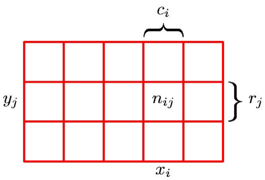


Example: having balls within two possible boxes and of two colors, one random variable denoted B represents the identity of the box, and can take two values (r or b). The ball color is another random variable, F, and can take the values g or o.

Joint probability : $p(X = x_i, Y = y_j) = \frac{n_{ij}}{N}$

Marginal Probability : $p(X = x_i) = \frac{c_i}{N}$

Conditional probability: $p(Y = y_j \mid X = x_i) = \frac{n_{ij}}{c_{ij}}$



Sum rule (marginalization) : $p(X = x_i) = \sum_{j=1}^{L} p(X = x_i, Y = y_j)$
which we can write

$$p(X) = \sum_{Y} p(X, Y)$$

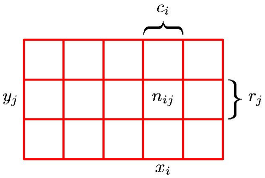
Product rule(conditioning) :
$p(X = x_i, Y = y_j) = p(Y = y_j \mid X = x_i)p(X = x_i)$

$$p(Y, X) = p(Y \mid X)p(X)$$

Joint probability : $p(X = x_i, Y = y_j) = \frac{n_{ij}}{N}$

Marginal Probability : $p(X = x_i) = \frac{c_i}{N}$

Conditional probability: $p(Y = y_j \mid X = x_i) = \frac{n_{ij}}{c_{ij}}$



Sum rule (marginalization) : $p(X = x_i) = \sum_{j=1}^{L} p(X = x_i, Y = y_j)$
which we can write

$$p(X) = \sum_{Y} p(X, Y)$$

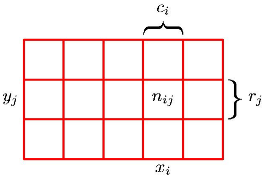
Product rule(conditioning) :
$p(X = x_i, Y = y_j) = p(Y = y_j \mid X = x_i)p(X = x_i)$

$$p(Y, X) = p(Y \mid X)p(X)$$

Joint probability : $p(X = x_i, Y = y_j) = \frac{n_{ij}}{N}$

Marginal Probability : $p(X = x_i) = \frac{c_i}{N}$

Conditional probability: $p(Y = y_j \mid X = x_i) = \frac{n_{ij}}{c_{ij}}$



Sum rule (marginalization) : $p(X = x_i) = \sum_{j=1}^{L} p(X = x_i, Y = y_j)$
which we can write

$$p(X) = \sum_{Y} p(X, Y)$$

Product rule(conditioning) :
$p(X = x_i, Y = y_j) = p(Y = y_j \mid X = x_i)p(X = x_i)$

$$p(Y, X) = p(Y \mid X)p(X)$$

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

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

$$P(X = x) = \sum_y P(X = x \mid Y = y) P(Y = y)$$

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

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

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

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

$$P(X = x) = \sum_y P(X = x \mid Y = y) P(Y = y)$$

posterior $\propto$ likelihood $\times$ prior

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

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

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

$$P(X = x) = \sum_y P(X = x \mid Y = y) P(Y = y)$$

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

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

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

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

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

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

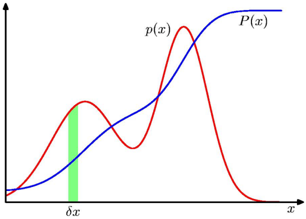
Note: if Y is finite, we can normalize numerically and we do not need $P(X = x)$.

Probability
Continuous variables

$p(x) \geq 0; \qquad \int_{-\infty}^{\infty} p(x)dx = 1$

$P(A) = \int_A p(x)dx$

$P(X \in [a, b]) = \int_a^b p(x)dx$



For continuous variables we have to define carefully the events. We usually consider **continuous** probability distributions, to which correspond **probability density functions** (denoted p).

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

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

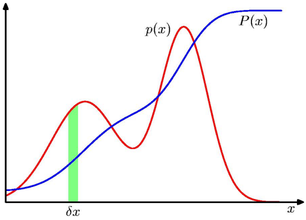
In $\mathbb{R}$, the cumulative distribution function (cdf) is defined as

$$P(X) = P(X \leq x) = \int_{-\infty}^{x} p(x)dx$$

$p(x) \geq 0; \qquad \int_{-\infty}^{\infty} p(x)dx = 1$

$P(A) = \int_A p(x)dx$

$P(X \in [a,b]) = \int_a^b p(x)dx$



For continuous variables we have to define carefully the events. We usually consider **continuous** probability distributions, to which correspond **probability density functions** (denoted p).

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

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

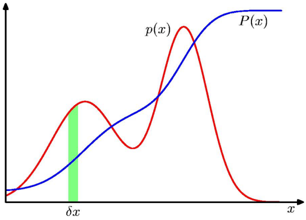
In $\mathbb{R}$, the cumulative distribution function (cdf) is defined as

$$P(X) = P(X \leq x) = \int_{-\infty}^{x} p(x)dx$$

$p(x) \geq 0; \qquad \int_{-\infty}^{\infty} p(x)dx = 1$

$P(A) = \int_A p(x)dx$

$P(X \in [a, b]) = \int_a^b p(x)dx$



For continuous variables we have to define carefully the events. We usually consider **continuous** probability distributions, to which correspond **probability density functions** (denoted p).

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

$$P(X) = P(X \leq x) = \int_{-\infty}^{x} p(x)dx$$

It is often of interest to computed weighted average of a function f, where the weights denote the probability of a given variables. This leads to the definition of expectations.

$$E(f) = \sum_x P(X = x)\, f(x) \qquad E(f) = \int_x p(x)\, f(x) dx$$

Particular case: if $f(x) = x$, the expectation corresponds to the mean of the probability distribution.

Similarly, we can define the conditional expectation

$$E_{X|Y}(f) = \sum_x P(X = x \mid Y)\, f(x).$$

Approximation of expectation. Given a set of variables drawn from $p(x)$

$$E(f) = \frac{1}{N} \sum_{n=1}^{N} f(x_n).$$

It is often of interest to computed weighted average of a function f, where the weights denote the probability of a given variables. This leads to the definition of expectations.

$$E(f) = \sum_x P(X = x)\, f(x) \qquad E(f) = \int_x p(x)\, f(x) dx$$

Particular case: if $f(x) = x$, the expectation corresponds to the mean of the probability distribution.

Similarly, we can define the conditional expectation

$$E_{X|Y}(f) = \sum_x P(X = x \mid Y)\, f(x).$$

Approximation of expectation. Given a set of variables drawn from $p(x)$

$$E(f) = \frac{1}{N} \sum_{n=1}^{N} f(x_n).$$

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

$$\forall x, y, P(X = x, Y = y) = P(X = x)P(Y = y).$$

Note: in general, 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

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

$$\forall x, y, P(X = x, Y = y) = P(X = x)P(Y = y).$$

Note: in general, 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.
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.

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

With a crude approximation of f near x_n

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

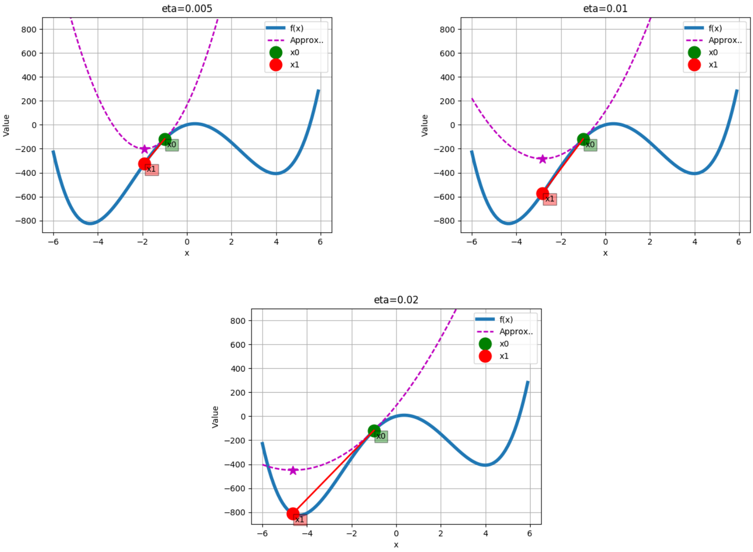
we get

$$\nabla \hat{f}(x) = \nabla f(x)^T + \frac{1}{\eta}(x - x_n)$$

so that

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

Gradient descent
Example



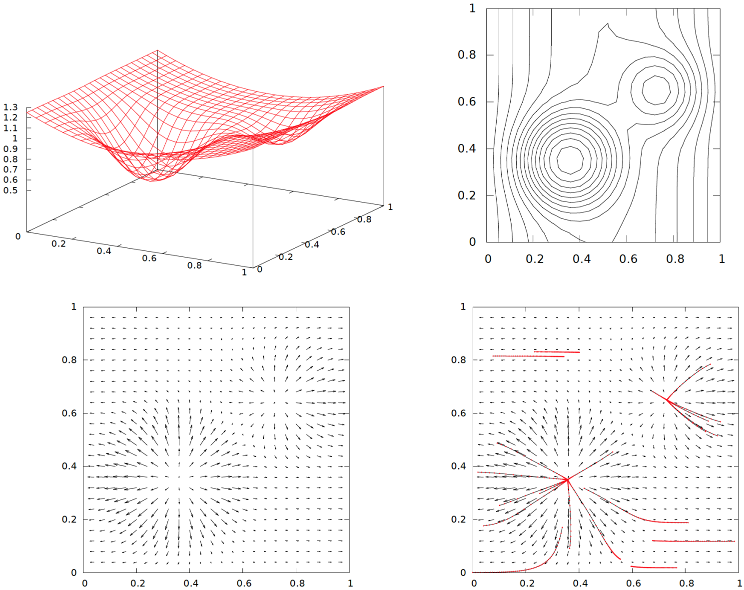
The resulting iterative rule takes the form of:

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

- finds a better x in the 'steepest descent' direction.
- only finds a **local** minimum
- choosing x_1 (as much as possible near optimal minimum) and η_n (too small: slow updates; too large: leave basin of attraction; oscillates around minimum) are critical

Gradient descent

Example in 2D



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

$$\eta_n = \operatorname*{argmin}_t f(x_n - t\nabla f(x_n))$$

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

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

by something more fitting to the problem at hand.

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

$$\eta_n = \operatorname*{argmin}_t f(x_n - t\nabla f(x_n))$$

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

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

by something more fitting to the problem at hand.

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

$$\eta_n = \operatorname*{argmin}_t f(x_n - t\nabla f(x_n))$$

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

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

by something more fitting to the problem at hand.

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

$$\eta_n = \operatorname*{argmin}_t f(x_n - t\nabla f(x_n))$$

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

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

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.

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**:

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

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.

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**:

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

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.

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**:

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

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

Let p be the true distribution of our data, and

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

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

There are three principal types of predictions:

- Classification
- Regression
- Density estimation

Let p be the true distribution of our data, and

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

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

There are three principal types of predictions:

- Classification
- Regression
- Density estimation

Classification

In that case, $Z = (X, Y)$, with typically $X \in \mathbb{R}^D$ and $Y \in \{1, \dots, C\}$.

One typically want to estimate $\operatorname{argmax}_y f_y(x)$

probabilistic case:

$$\operatorname{argmax}_y P(Y = y \mid X = x)$$

Examples: object recognition, cancer detection, speech processing.

Note: X could be a mix of dicrete and continuous data (structured input)

Classification

In that case, $Z = (X, Y)$, with typically $X \in \mathbb{R}^D$ and $Y \in \{1, \ldots, C\}$.

One typically want to estimate $\operatorname{argmax}_y f_y(x)$

probabilistic case:

$$\operatorname{argmax}_y P(Y = y \mid X = x)$$

Examples: object recognition, cancer detection, speech processing.

Note: X could be a mix of dicrete and continuous data (structured input)

Regression

In that case, $Z = (X, Y)$, with typically $X \in \mathbb{R}^D$ and $Y \in \mathbb{R}$.

One typically want to estimate a functional of the input $y = f(x)$

Probabilistic case: estimate the expected value

$$E(Y \mid X = x)$$

Examples: customer satisfaction, stock prediction, epidemiology.

Density estimation

$\mathcal{Z} = \mathbb{R}^D$

We want to estimate $p(z)$

Data visualization, pre-processing, outlier detection.

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

- For classification:

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

- For regression:

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

- For density estimation:

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

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

- For classification:

$$f(x; w_1, \ldots, w_C) = \operatorname*{argmax}_y \langle w_y, x \rangle$$

- For regression:

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

- For density estimation:

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

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

- For classification:

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

- For regression:

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

- For density estimation:

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

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:

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

We define the “good” functionals through a loss function

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

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:

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

Note: the loss may include other terms related to f itself (eg. for regularization, cf later and lab)

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:

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

Note: the loss may include other terms related to f itself (eg. for regularization, cf later and lab)

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:

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

Note: the loss may include other terms related to f itself (eg. for regularization, cf later and lab)

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:

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

Note: the loss may include other terms related to f itself (eg. for regularization, cf later and lab)

We are looking for an f with a small expected risk

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

which means that our learning procedure should pick

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

Unfortunately this quantity is not available. Instead, we can use the set of training samples $\mathcal{D} = \{Z_1, \ldots, Z_n\}$ supposed to be i.i.d to compute an estimate of it called 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 are looking for an f with a small expected risk

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

which means that our learning procedure should pick

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

Unfortunately this quantity is not available. Instead, we can use the set of training samples $\mathcal{D} = \{Z_1, \dots, Z_n\}$ supposed to be i.i.d to compute an estimate of it called 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

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

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

We have

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

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 $\mathcal{F}$.

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 $\mathcal{F}$.

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

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

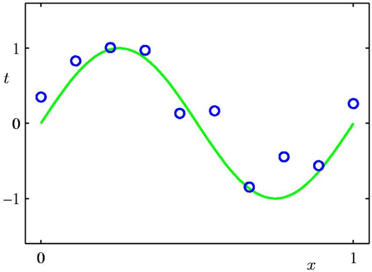
- Can we bound $R(f)$ with $\hat{R}(f, \mathcal{D})$?

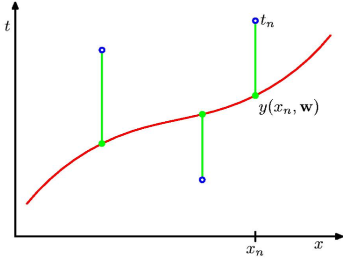
  Yes if f is not chosen using $\mathcal{D}$. Since the Z_n are independent, we just need to take into account the variance of $\hat{R}(f, \mathcal{D})$.

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

  Unfortunately, not simply, and not without additional information about $\mathcal{F}$.

Given a set of noisy data points coming from an underlying model, find a polynomial that best fit the data





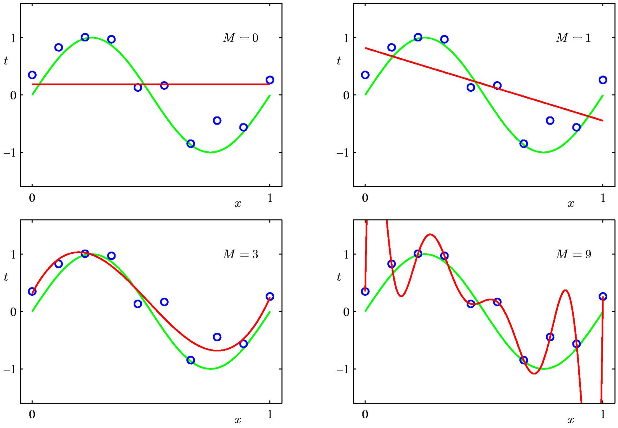
Model (polynomial):

$$y(x, \mathrm{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Error function (empirical risk):

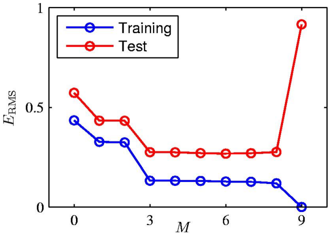
$$E(\mathrm{w}) = \frac{1}{2} \sum_{n=1}^{N} \Big(y(x_n, \mathrm{w}) - t_n \Big)^2$$

We fix the training datasets $\mathcal{D}$ and increase the space $\mathcal{F}$
$0^{th}, 1^{st}, 3^{rd}, 9^{th}$ order polynomial



Test performance: measure the error of the model on independent points drawn from the underlying function

Under and over fitting



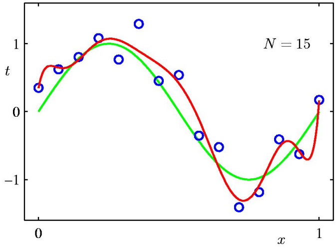
Fitted coefficients

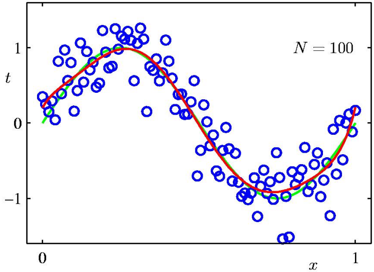
	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Root mean square error $E_{RMS} = \sqrt{2E(\mathrm{w}^*)/N}$

How to avoid the overfitting problem?

We fix the model complexity and increase the size of the data $\mathcal{D}$



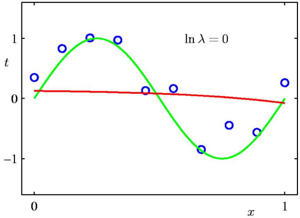


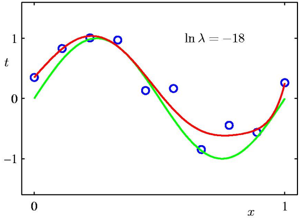
9^{th} order polynomial

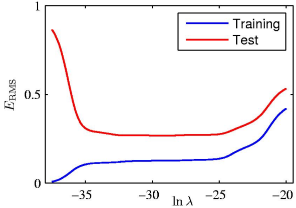
Increasing the dataset size reduces the effect of overfitting

Penalize large coefficients

$$\tilde{E}(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(y(x_n,\mathrm{w}) - t_n\right)^2 + \frac{\lambda}{2}\sum_j w_j^2$$







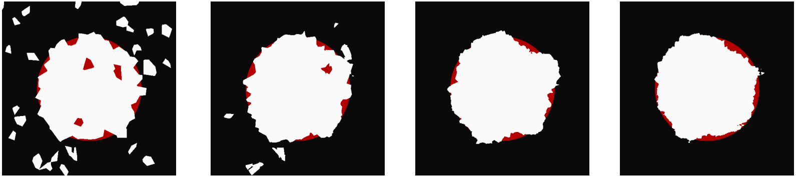
	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

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

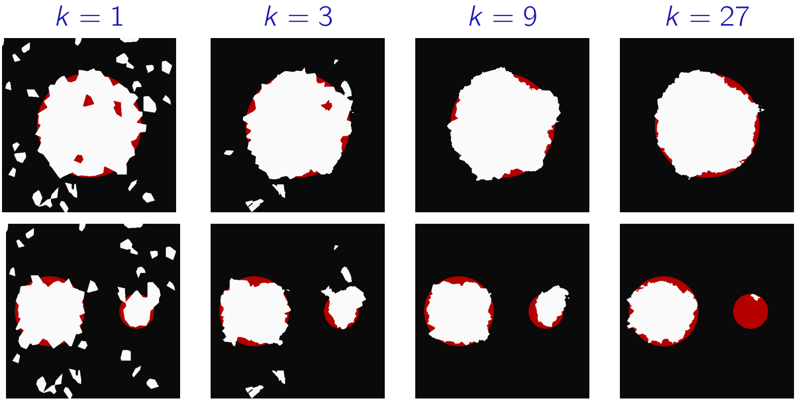
Under and Over-fitting

Example: k-Nearest Neighbors (cont.)

$k = 1$ $\quad$ $k = 3$ $\quad$ $k = 9$ $\quad$ $k = 27$



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



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 $\mathcal{F}$. Thus, intuitively, it reflects the ability to model any arbitrary functional.

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 $\mathcal{F}$.

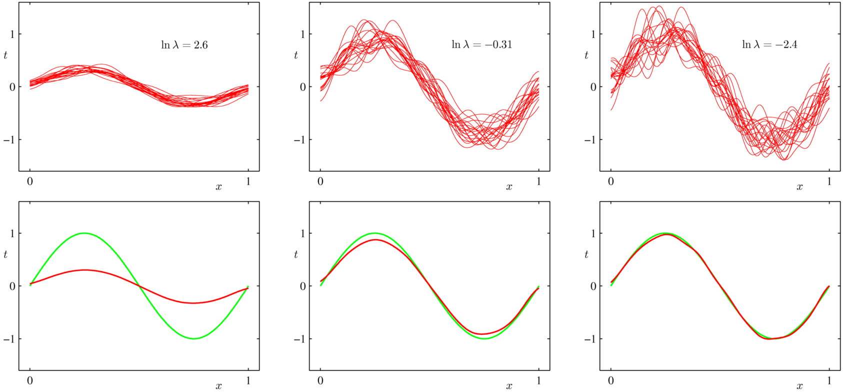
In classification, we can consider the capacity of $\mathcal{F}$. which is the maximum size of a set which can be arbitrarily labelled by a function from $\mathcal{F}$. Thus, intuitively, it reflects the ability to model any arbitrary functional.

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 $\mathcal{F}$. Thus, intuitively, it reflects the ability to model any arbitrary functional.

Example. Take 25 random datasets to do the polynomial fitting, varying the degree of regularization. What do we observe?



We can decompose the expected error as the sum of a bias and a variance term. For a given x, if $f(x)$ denotes the 'true' prediction, and $f^*(x)$ the prediction using a training dataset,

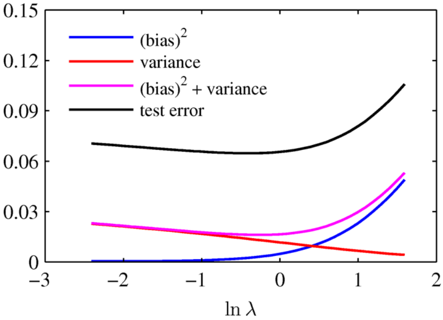
$$
\begin{aligned}
& E_{\mathcal{D}}\left((f^*(x) - f(x))^2\right) \qquad (1) \\
=& E_{\mathcal{D}}\Big((f^*(x))^2\Big) - 2E_{\mathcal{D}}(f^*(x))f(x) + f^2(x) \\
=& \Big(E_{\mathcal{D}}((f^*(x))^2) - E_{\mathcal{D}}(f^*(x))^2\Big) \\
& + \Big(E_{\mathcal{D}}(f^*(x))^2 - 2E_{\mathcal{D}}(f^*(x))f(x) + f^2(x)\Big) \\
& = \underbrace{V_{\mathcal{D}}(f^*(x))}_{\text{Variance}} + \underbrace{\Big(E_{\mathcal{D}}(f^*(x)) - f(x)\Big)^2}_{\text{Bias}}
\end{aligned}
$$

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.

We can decompose the expected error as the sum of a bias and a variance term. For a given x, if $f(x)$ denotes the 'true' prediction, and $f^*(x)$ the prediction using a training dataset,

$$
\begin{aligned}
& E_{\mathcal{D}}\left((f^*(x) - f(x))^2\right) \qquad (1) \\
=& E_{\mathcal{D}}\Big((f^*(x))^2\Big) - 2E_{\mathcal{D}}(f^*(x))f(x) + f^2(x) \\
=& \Big(E_{\mathcal{D}}((f^*(x))^2) - E_{\mathcal{D}}(f^*(x))^2\Big) \\
& + \Big(E_{\mathcal{D}}(f^*(x))^2 - 2E_{\mathcal{D}}(f^*(x))f(x) + f^2(x)\Big) \\
& = \underbrace{V_{\mathcal{D}}(f^*(x))}_{\text{Variance}} + \underbrace{\Big(E_{\mathcal{D}}(f^*(x)) - f(x)\Big)^2}_{\text{Bias}}
\end{aligned}
$$

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.



An over-regularized model (small capacity) will have a large bias, whereas an under-regularized model (large capacity) will a large variance.

The main strategies to control over-fitting is to increase the amount of data, or through some form of 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 $\mathrm{argmin}_{f \in \mathcal{F}} \hat{R}(f; \mathcal{D}) + H(f)$

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.

A machine learning algorithm combines

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

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.

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.

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.

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

Models have parameters to be trained, and often involve several meta-parameters that need to be set (eg degree of polynomial, regularization parameter λ).
The ideal development cycle is

$$\boxed{\text{Write code}} \longrightarrow \boxed{\text{Train}}$$

Models have parameters to be trained, and often involve several meta-parameters that need to be set (eg degree of polynomial, regularization parameter λ).
The ideal development cycle is

$$\boxed{\text{Write code}} \longrightarrow \boxed{\text{Train}} \longrightarrow \boxed{\text{Test}}$$

Models have parameters to be trained, and often involve several meta-parameters that need to be set (eg degree of polynomial, regularization parameter λ).
The ideal development cycle is

Write code $\longrightarrow$ Train $\longrightarrow$ Test $\longrightarrow$ Paper

Machine learning in practice
Cheating by over-fitting

Models have parameters to be trained, and often involve several meta-parameters that need to be set (eg degree of polynomial, regularization parameter λ).
The ideal development cycle is

$$\begin{tikzcd} \text{Write code} \arrow[r] & \text{Train} \arrow[r] & \text{Test} \arrow[r] & \text{Paper} \end{tikzcd}$$

or in practice something like

$$\begin{tikzcd} \text{Write code} \arrow[r] & \text{Train} \arrow[r] \arrow[l, bend left] & \text{Test} \arrow[r] & \text{Paper} \end{tikzcd}$$

Machine learning in practice
Cheating by over-fitting

Models have parameters to be trained, and often involve several meta-parameters that need to be set (eg degree of polynomial, regularization parameter λ).
The ideal development cycle is

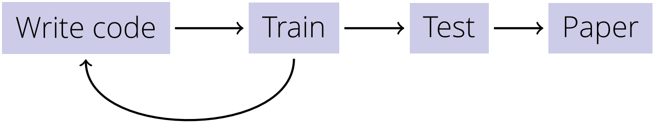
$$\boxed{\text{Write code}} \longrightarrow \boxed{\text{Train}} \longrightarrow \boxed{\text{Test}} \longrightarrow \boxed{\text{Paper}}$$

or in practice something like

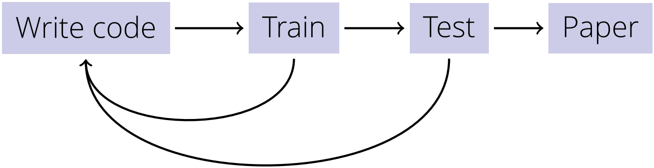
$$\begin{tikzcd}
\boxed{\text{Write code}} \arrow[r] & \boxed{\text{Train}} \arrow[l, bend left] \arrow[r] & \boxed{\text{Test}} \arrow[r] & \boxed{\text{Paper}}
\end{tikzcd}$$

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

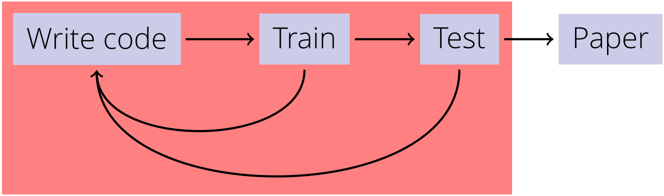
Unfortunately, it often looks like



Unfortunately, it often looks like

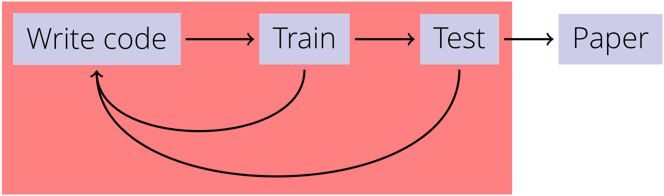


Unfortunately, it often looks like



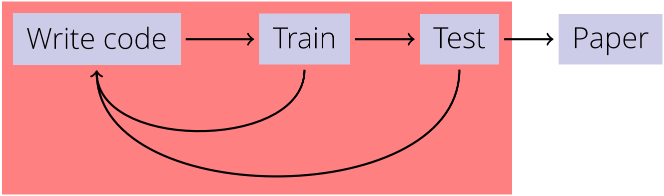
Machine learning in practice
Cheating by over-fitting (cont.)

Unfortunately, it often looks like

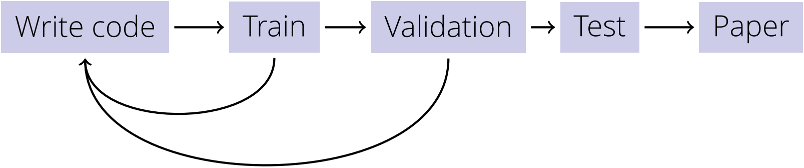


This should be avoided at all costs. The standard strategy is to have a separate validation set for the tuning.

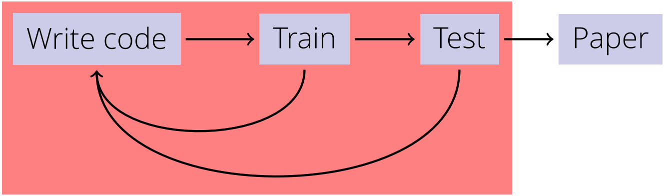
Unfortunately, it often looks like



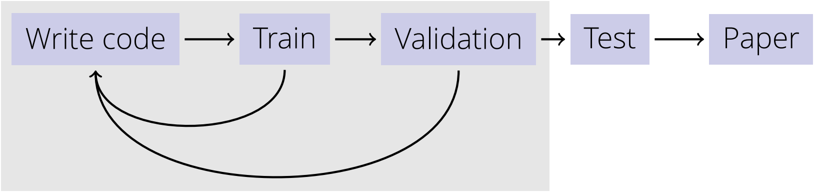
This should be avoided at all costs. The standard strategy is to have a separate validation set for the tuning.



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).

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

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

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

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

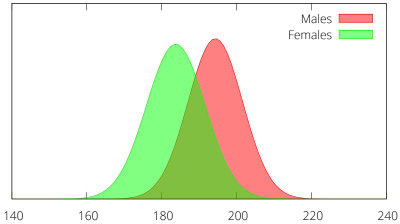
The generative approaches rely on a model of the data, even if it is not the quantity of interest.

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

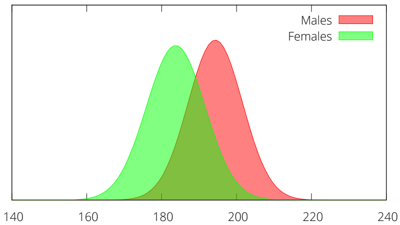
Females		**Males**	
190	180	190	195
182	193	193	184
188	179	199	190
184	186	200	203
196	185	192	205
173	169	190	201

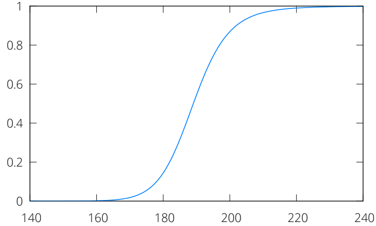
Discriminative vs. generative
Example: Gender prediction (cont.)

We can either model $P(H \mid G)$

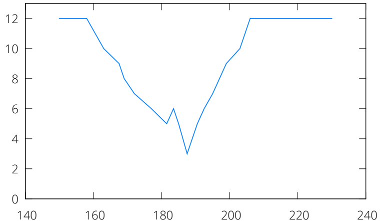


We can either model $P(H \mid G)$ and from that derive $P(G \mid H)$ using Bayes' law.





But we can also directly look for the best threshold:



- Supervised vs. unsupervised

  Supervised learning has access to the values to predict.

  Unsupervised methods don't. They are often used in density estimation

  Semi-supervised: the value to predict is available for some data points but not for others

- 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.

- Supervised vs. unsupervised

  Supervised learning has access to the values to predict.

  Unsupervised methods don't. They are often used in density estimation

  Semi-supervised: the value to predict is available for some data points but not for others

- 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.

Relation to other fields

- 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

Practical session: observing the overfitting and underfitting issues with polynomial regression.

Form: jupyter notebook, with parts to fill in. Use scikit-learn.

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

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

Practical session: observing the overfitting and underfitting issues with polynomial regression.

Form: jupyter notebook, with parts to fill in. Use scikit-learn.

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

$$\operatorname*{argmin}_{\alpha_1, \dots, \alpha_K} \; L(\alpha_1, \dots, \alpha_K) \,?$$

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

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

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 in the room CO 5 **at 13:15-15:00**.

Instructions on how to download the lab will be given then.