# Machine Learning

Marc Sebban & Amaury Habrard
LaHC

## Contents

## 1 What is Machine Learning?

Machine Learning aims at knowing how to make algorithms that can *learn* from data. They are divided in two category:

- Supervised learning, subdivided into
  - Classification: predict a yes/no answer
  - Regression: predict a continuous value, such as the price of a house
  - Ranking: output the "most relevant" data

  The aim is to predict fro labelled data
- Unsupervised learning, subdivided into
  - Clustering
  - Dimensionality Reduction

  The aim is to find the underlying structure of unlabelled data

**Possible Applications** Computer Vision, Robotics, Speech Recognition, Artificial Intelligence

**Required Skills**

- Convex Optimization
- Algorithm: Asymptotic behaviour

We will mainly use SVM (Support Vector Machine), that deals with classification problems. They use the *kernel trick*, which is projection of the data on a high-dimensional space (potentially infinite) where the data becomes linearly separable.

## 2 Supervised learning problem

### 2.1 Notations

- Let $S = \{z_i = (\mathbf{x_i}, y_i)\}_{i=1}^m$ be a set of $m$ training examples i.i.d. from an unknown joint distribution $\mathcal{D}_\mathcal{Z}$ over a space $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$
- The $\mathbf{x_i}$ values ($\mathbf{x_i} \in \mathcal{X}$) are typically vectors in $\mathbb{R}^d$ whose components are usually called *features.*
- The $y$ values ($y \in \mathcal{Y}$) are drawn from a discrete set of *classes/labels* (typically $\mathcal{Y} = \{-1, +1\}$ in *binary classification*) or are continuous values (*regression*)
- We assume that there exists a *target function* $f$ such that $y = f(\mathbf{x})$, $\forall(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$.

**Definition 1.** *A supervised learning algorithm $L$ automatically outputs from $S$ a model or a classifier (or a hypothesis) $h \in \mathcal{H}$ as close to $f$ as possible.*

## 2.2 Curse of dimensionality - Overfitting - Underfitting

The number of training example is very important! Sadly, as the number of features or dimension grows, the amount of data (i.e. examples necessary to learn) grows exponentially: it is the *curse of dimensionality.*

To avoid this problem, we can:

- pre-process the data into a lower dimensional space
- regularize the underlying optimization problem at running time

This issue is very closed to *overfitting.*

**Definition 2** (Overfitting). *In statistics,* overfitting *occurs when a model is excessively complex, such as having too many degrees of freedom (e.g. polynomial of high order) with respect to the amount of data available $\rightarrow$ use a* regularization.

**Definition 3** (Underfitting). Underfitting *occurs when a statistical model or machine learning algorithm cannot capture the underlying trend of the data.*

To pick the best hypothesis $h^*$, we need a criterion to assess the quality of $h$. Given a non-negative loss function $\updownarrow : \mathcal{H} \times \mathcal{Z} \rightarrow \mathbb{R}^+$ measuring the degree of agreement between $h(\mathbf{x})$ and $y$, we can define the *tree risk.*

**Definition 4** (True Risk). *The true risk $\mathcal{R}^\ell(h)$ (also called* generalization error*) of a hypothesis $h$ with respect to a loss function $\ell$ corresponds to the expected loss suffered by $h$ over the distribution $\mathcal{D}_\mathcal{Z}$.*

$$\mathcal{R}^\ell(h) = \mathbb{E}_{\mathcal{Z}\sim\mathcal{D}_\mathcal{Z}}\ell(h, z)$$

Unfortunately, $\mathcal{R}^\ell(h)$ cannot be computed as $\mathcal{D}_\mathcal{Z}$ is unknown, so we try to minimise the *empirical risk* $\hat{\mathcal{R}}^\ell$, a statistical measure of the true risk over $S$.

**Definition 5** (Empirical Risk). *Let $S = \{z_i = (\mathbf{x_i}, y_i))\}_{i=1}^m$ be a training sample. The empirical risk $\hat{\mathcal{R}}^\ell$ (also called empirical error) of a hypothesis $h \in \mathcal{H}$ with respect to a loss function ' corresponds to the expected loss suffered by $h$ on $S$.*

$$\hat{\mathcal{R}}_\ell(h) = \frac{1}{m}\sum_{i=1}^{m}\ell(h, z_i)$$

**Definition 6** (0/1 loss). *The most natural loss function for binary classification is the 0/1 loss (also called classification error)*

$$\ell_{0/1}(h, z) = 1 \qquad \textit{if } yh(x) < 0 \textit{ and 0 otherwise}$$

*$\mathcal{R}^{\ell_{0/1}}$ then corresponds to the proportion of correct predictions.*

**Warning** Due to the non convexity and non differentiability of the 0/1 loss, minimizing the empirical risk is NP-hard. For this reason, we use surrogate loss functions such that:

- *Hinge loss* (used in SVM): $\ell_{hinge}(h, z) = \max(0, 1 - yh(x))$
- *Exponential loss* (used in boosting): $\ell_{exp}(h, z) = e^{-yh(x)}$
- *Logistic loss* (used in logistic regression): $\ell_{log}(h, z) = \ln(1 + e^{-yh(x)})$

## 2.3 Regularized Risk Minimization

To prevent the algorithm from overfitting, a supervised learning problem often take the following regularized form:

$$\min_{h \in \mathcal{H}} \hat{\mathcal{R}}^{\ell}(h) + \lambda ||h||_p$$

Where $\lambda$ is a constant penalizing "too complex" models, and $||.||_p$ a $\ell_p$-norm over the classifier $h$.

**Definition 7** ($\ell_p$-norm)**.** *If $\theta$ is a d-dimensional vector:*

$$||\theta||_p = \left( \sum_{i=1}^{d} |\theta_i|^p \right)^{\frac{1}{p}}$$

The $\ell_2$-norm is used to reduce the risk of overfitting (it decreases the large values of the model), and the $\ell_1$ also allows the induction of sparse models - i.e. with less features (example: LASSO or $\ell_1$-SVM).

**Remark** Increasing $\theta$ with the $\ell_1$-norm causes more and more of the parameters $\theta_j$ to be driven to zero. The gradient on the $\ell_1$-norm is constant w.r.t. the magnitude of each vector component.

**Downside** The $l_1$-norm is not differentiable.

## 2.4 Bias/Variance trade-of

There are three sources of error between $h \in \mathcal{H}$ and the target function $f \in \mathcal{F}$:

1. The inductive bias: nothing guarantees the equality between the target concept space $\mathcal{F}$ and the selected class of hypotheses $\mathcal{H}$, even if the learner is able to provide an optimal hypothesis $h^*$ from $\mathcal{H}$.
2. The variance: since the training set $S$ is finite and randomly drawn from $\mathcal{D}_{\mathcal{Z}}$, the learner usually does not provide the optimal hypothesis $h^*$.
3. The presence of noise: some training examples can be mislabelled. The learner receives a training set of a "noisy" function $f_b = f + \varepsilon$.

The Bias/Variance trade-off comes from the Mean Square Error (MSE), in statistics:

**Definition 8** (MSE)**.** *Let $\theta$ a theoretical parameter ($\mathcal{R}(h)$ in our case) and $\hat{\theta}$ an estimate of $\theta$ ($\hat{\mathcal{R}}(h)$ in our case). Let $B = \mathbb{E}(\theta) - \theta$ be the bias of $\hat{\theta}$ w.r.t. $\theta$. The MSE assesses the quality of $\theta$ in terms of its variation and unbiasedness. It is the expected value of the square loss between $\hat{\theta}$ and $\theta$.*

$$\begin{aligned} MSE &= \mathbb{E}_z[(\hat{\theta} - \theta)^2] \\ &= \mathbb{E}_z[(\hat{\theta} - \mathbb{E}(\hat{\theta}) + \mathbb{E}(\hat{\theta}) - \theta)^2] \\ &= \mathbb{E}_z[(\hat{\theta} - \mathbb{E}(\hat{\theta}) + B)^2] \\ &= \mathbb{V}(\hat{\theta}) + B^2 \end{aligned}$$

## 2.5 Statistical learning theory

**Definition 9** (Empirical Risk Minimization)**.** *The ERM principle rests on the fact that if h works well on the training set S it might also work well on new examples.*

**Definition 10** (Probably Approximately Correct (PAC) Condition). *[Valiant 1984] The ERM principle is valid if the true risk of the hypothesis $h \in \mathcal{H}$ induced from $S$ is closed to the true risk of the optimal hypothesis $h^* \in \mathcal{H}$*

$$h = \arg\min_{h_i \in \mathcal{H}} \hat{\mathcal{R}}(h_i)$$
$$h^* = \arg\min_{h_i \in \mathcal{H}} \mathcal{R}(h_i)$$

*Condition of validity of the ERM principle:*

$$\forall \mathcal{D}_{\mathcal{Z}}, \forall \gamma \geq 1, \forall \delta \leq 1, \mathbb{P}(|\mathcal{R}(h) = \mathcal{R}(h^*)| \geq \gamma) \leq \delta$$

**Definition 11** (Bayesian error). *The bayesian error $\epsilon^*$ is the lowest possible error rate (or irreducible error) for any hypothesis $h$.*

$$\epsilon^8 = \int_{x \in R_i \ s.t. \ y \neq C_i} \mathbb{P}(C_i|x)\mathbb{P}(x)dx$$

*where $x$ is an instance, $y$ its corresponding label, $R_i$ is the area/region that a classifier function $h$ classifies as $C_i$.*

**Remark** In many application, $\epsilon^* > 0$, and as $S$ is finite, selecting the optimal $h$ does not imply getting the optimal hypothesis $h^*$.

The law of large numbers prompts us to increase the size of the learning set and to search for the minimal size m that allows us to fulfil the PAC condition:

$$\forall \mathcal{D}_{\mathcal{Z}}, \forall \gamma \geq 0, \forall \delta \leq 1, \exists m \text{ s.t. } \mathbb{P}(|\mathcal{R}(h_m) - \mathcal{R}(h^*)| \geq \gamma) \leq \delta$$

Where $h_m$ is the hypothesis learnt from a training set of size $m$.

**Question** What is the conditions on the *minimum number of required examples* for the empirical risk and the true risk of the induced hypothesis $h$ to converge towards the true risk of $h^*$ ?
We will differentiate $|\mathcal{H}|$ finite and $|\mathcal{H}|$ infinite.

### 2.5.1 When $|\mathcal{H}|$ is finite

### 2.5.2 When $|\mathcal{H}|$ is infinite

**Remarks** On the VC theory.

- The only property that matters is the *size of the hypothesis space* and not *how the algorithm searches the space*
- Therefore, the VC theory is meaningful when the learning algorithm performs minimization of $\hat{\text{R}}(h)$ in the *full hypothesis space.*
- It is *useless for local algorithms*, like the $k$-NM which has an infinite $d_{\mathcal{H}}$.
- Two analytical frameworks to take into account the algorithm $L$ to derive generalization bounds: *Uniform stability* and Algorithmic robustness. The goal is to bound:

  $$\mathbb{P}(|\mathcal{R}(L, h_S) - \hat{\mathcal{R}}(L, h_S)| \geq \gamma)$$

  which differs from what we studied before:

  $$\mathbb{P}(\sup_{h \in \mathcal{H}} |\mathcal{R}(h) - \hat{\mathcal{R}}(h)| \geq \gamma)$$

## 2.6 Uniform stability

We only focus here on Uniform Stability. Variance versus Stability Statistical learning theory prompts us to reduce the variance without altering the bias. Having a low variance is equivalent to having high stability. How to relate the generalization error Rh to the stability of an algorithm L which induces h? Intuitively, an algorithm L is said stable if it is robust to small changes in the training sample, i.e., the variation in its output h is small.

Given a training set $S$ of size $m$, we build $\forall i = 1...m$:

- $S^{\setminus i} = \{z_1, ..., z_{i-1}, z_{i+1}, ..., z_m\}$ by removing the $i$-th element of $S$.
- $S^i = \{z_1, ..., z_{i-1}, z'_i, z_{i+1}, ..., z_m\}$ by replacing the $i$-th element by $z'_i$ drawn i.i.d. from $\mathcal{D}_\mathcal{Z}$.

**Definition 12** (Uniform stability)**.** *[Bousquet and Elisseef][2002] An algorithm L has uniform stability $\frac{\beta}{m}$ with respect to a loss function $\ell$ if the following holds:*

$$\forall S, \forall i \in \{1, ..., m\}, \sup_z |\ell(h_S, z) - \ell(h_{S^{\setminus i}}, z)| \leq \frac{\beta}{m}$$

*Where $\beta$ is a positive constant, $h_S$ and $h_{S^{\setminus i}}$ are the hypothesis learned by L from $S$ and $S^{\setminus i}$ respectively.*

**Propriety 1** (Generalisation bound using uniform stability)**.** *Let $S$ be a training sample of size $m$ and $\delta > 0$. For any algorithm L with uniform stability $\frac{\beta}{m}$ with respect to a loss function $\ell$ bounded by $M$, with probability $1 - \delta$, we have:*

$$\mathcal{R}_{h_S} \leq \hat{\mathcal{R}}_{h_S} + \frac{2\beta}{m} + (4\beta + M)\sqrt{\frac{\ln \frac{1}{\delta}}{2m}}$$

**Theorem 1** (Kearns and Ron)**.** *[1999] An algorithm L having an hypothesis space of finite VC-dimension is stable in the sens that its stability is bounded by ots VC-dimension.*

**Corollary 1.** *Using the stability as a complexity measure does not give worse bounds than using the VC-dimension.*

*Proof.* Of Generalisation Bound.
The proof is based on McDiarmid's Theorem (1989).

**Theorem 2.** *[McDiarmid][1989] Let $S$ and $S^i$ defined as above, let $F : \mathcal{Z}^m \to \mathbb{R}$ be a function for which there exists constants $c_i (i = 1, ..., m)$ such that*

$$\sup_{s \in \mathcal{Z}^m, z'_i \in \mathcal{Z}} |F(S) - F(S^i)| \leq c_i$$

*then*

$$P_S(F(S) - \mathbb{E}[F(S)] \geq \gamma) \leq e^{-2\gamma^2 / \sum_{i=1}^m c_i^2}$$

Let's show that $F(S) = \mathcal{R}_{h_S} - \hat{\mathcal{R}}_{h_S}$ (for the sake of simplicity $F(S) = \mathcal{R} - \hat{\mathcal{R}}$) and $F(S^i) = \mathcal{R}^i - \hat{\mathcal{R}}^i$ satisfy the previous condition.

See more details in the slides □

**How to find the stability constant?** When the learning problem takes the following form:

$$\min_{h \in \mathcal{H}} \frac{1}{m} \sum_i \ell(h, z_i) + \lambda ||h||^2_\mathcal{F}$$

we can show [Bousquet and Elisseef 2002] that the stability constant is defined as follows:

$$\beta \leq \frac{\sigma^2}{2\lambda}$$

where $\sigma$ comes from the $\sigma$ admissibility of $\ell(h, z) = c(h(x), y)$ where $c$ is the associated cost function.

**Definition 13** ($\sigma$-admissibility)**.** *A loss function $\ell$ is $\sigma$-admissible if the associated cost function $c(h(x), y)$ is convex with respect to its first argument and the following condition holds:*

$$\forall y_1, y_2 \in \mathcal{Z}, \forall y' \in \mathcal{Y}, |c(y_1, y') - c(y_2, y')| \leq \sigma |y_1 - y_2|$$

**Examples:** See tutorial.

## 2.7 Model Selection

We saw that there is pften a trade-off between the bias and variance: it is important not to choose a hypothesis that is either too simple (underfitting) or too complex (overfitting). Model selection algorithms provide a method that automatically makes the trade-off between bias and variance.

#### Examples

1. Linear regression: What degree of the polynomial do you need to select?
2. $k$ nearest-neighbours: What is the right number $k$ of neighbours?

#### Model Selection

How to choose the best hypothesis in $M$ ($M = \{h_1, h_2, ...\}$)

- Bad Idea: choose the onbe with the lowest training error $\hat{\mathcal{R}}(h)$ (risk of overfitting)
- Good Idea: Hold-out $k$ cross-validation

```
  Input: A learning algorithm $L$ and a learning set $S$ of $m$ examples
  Output: An estimate $\hat{\mathcal{R}}'(h)$
1 Split $S$ randomly in $k$ subsets $S_1, ..., S_k$ (if $k = m$, leave-one-out CV);
2 for $i$=1 to $k$ do
3     Run $L$ on $S - S_i$ and induce classifier $h_i$;
4 end
5 Deduce the estimate $\hat{\mathcal{R}}'(h)$ of the true risk s.t. $\hat{\mathcal{R}}'(h) = \frac{1}{k}\sum_{i=1}^{k} \hat{\mathcal{R}}(h_i)$ where $\hat{\mathcal{R}}(h_i)$ is the error of $h_i$
    on $S_i$;
```

**Algorithm 1:** Hold-out $k$ cross-validation algorithm

# 3 The Regression Problem

## 3.1 Introduction

**The Regression Problem** How to optimize the parameter of the hyperplane

$$h_\theta(x) = \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_0$$

which fits the best the following set of training example $S$ ?

**Notations**

- $m$ is the number of training examples
- $x \in \mathbb{R}^n$ is the input feature vector of $n$ variables.
- $y$ is the output variable/target
- $(x, y)$ is the training example
- $(x^{(i)}, y^{(i)})$ is the $i$-th training example
- $h$ is the hypothesis that maps from input $x$ to output $y$.
- $h(x)$ is of the linear form: $h(x) = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n$. For conciseness, we define $x_0 = 1$ so that

$$h(x) = h_\theta(x) = \sum_{i=0}^{n} \theta^i x_i = \theta^T x$$

  Where $\theta = (\theta_0, ..., \theta_n) \in \mathbb{R}^{n+1}$.

We aims at choosing the parameters $\theta$ so that the hypothesis $h$ will make accurate predictions.

**Definition 14** (Non regularized Least Squares Problem)**.**

$$\min_\theta J(\theta) = \min_\theta \frac{1}{2m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

## 3.2 Learning Algorithms

There exist three main solutions for minimizing $J(\theta)$:

- Batch Gradient Descent
- Stochastic Gradient Descent
- Closed-form solution

**Basic Idea** Let us assume that $J(\theta)$ is differentiable:

- Start with some initialization of $\theta$ (e.g. $\vec{\theta} = 0$ or some randomly chosen vector.)
- Update$\theta$'s values so that to reduce $J(\theta)$) (by computing partial derivatives of $J(\theta)$ w.r.t. $\theta$).
- Repeat the process till convergence to the minimum of $J(\theta)$

**Update Rule**

**Remark** Note that with a slightly different initial starting point, you can end up to a completely different optimum. Fortunately,$J(\theta)$ is a *quadratic function* with only *one global optimum.*

**Update of the $i^{th}$ parameter of $\theta$** Cf slides for calculus We get the following update rule:

$$\theta_i := \theta_i - \alpha(h_\theta(x) - y)x_i$$

**Remark** If $m$ is huge, then the batch gradient descent will perform at each step a huge summation, therefore we need another algorithm.

```
1 Initialized $\vec{\theta}$;
2 repeat
3     $\forall\theta_i$ of $\theta$, $\theta_i := \theta_i - \alpha\frac{1}{m}\sum_{j=1}^{m}(h_\theta(x) - y)x_i$
4 until convergence (i.e. "stabilisation" of $J(\theta)$);
```

**Algorithm 2:** Batch Gradient Descending Algorithm

```
1 Initialized $\vec{\theta}$;
2 repeat
3     for j=1 to m do
4         $\forall\theta_i$ of $\theta$, $\theta_i := \theta_i - \alpha(h_\theta(x^{(j)}) - y^{(j)})x_i^{(j)}$
5     end
6 until convergence (i.e. "stabilisation" of $J(\theta)$);
```

**Algorithm 3:** Stochastic Descending Algorithm

**Remarks** The Stochastic Gradient Descent is much faster, and start each update of $\theta$ with the first training example, then the second, ... . However, it won't converge exactly to the global minimum, even though it tends to wander around some regions close to the global minimum.

**Influence of the learning rate $\alpha$** For some specific examples, $J(\theta)$ may increase. This may occur in some situations where $\theta$ is large ("zigzag" effect). To prevent the parameters $\theta$'s from oscillating around the global minimum we can take a smaller learning rate.

In most implementations, the learning rate is held constant. However, if you want to converge to "a minimum" you can slowly decrease $\alpha$ over time, such that at iteration i we get:

$$\alpha_i = \frac{C_1}{i + C_2}$$

which means you're guaranteed to converge "somewhere".

**Mini-batch gradient descent** A compromise between a batch gradient descent and a stochastic gradient descent, is to compute the gradient only on a (randomly selected) bunch of training examples (called a "*mini-batch*") at each step.

It may result in smoother convergence, as the gradient computed at each step uses more training examples than in the stochastic setting.

**Closed-form Solution** We use massively the trace and its proprieties (Cf slides for details). The optimal solution, $\nabla_\theta J(\theta) \overset{\text{set}}{=} \vec{0}$ boils down to

$$\theta = (X^T X)^{-1} X^T y$$

Where

$$X = \begin{pmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(n)})^T \end{pmatrix}$$

**Issues:** $(X^T X)^{-1}$ can be not invertible:

- Redundant features (linearly dependent) $\rightarrow$ perform a PCA (Principal Component Analysis)
- Too many features $\rightarrow$ delete some irrelevant features (feature selection algorithm)

**Remarks** For complexity reasons, we prefer to use the gradient descent rather than the normal equation when $n$ is height.

## 3.3 Objective function in linear regression

The objective function used is

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

**Definition 15** (Likelihood)**.** *Let $x_1, x_2, ...x_m$ be a random set of $m$ i.i.d. observations, coming from an unknown*

So we use the quadratic loss because it is equivalent to maximising the likelihood assuming that there are i.i.d gaussian errors on the data

## 3.4 Limitations of Linear Regression

Until now, we assumed that $y \in \mathbb{R}$ (continuous variable). Let us now assume that $y \in \{0, 1\}$ (discrete variable).

$$\text{predicting } y \simeq \text{ classification task}$$

Sometimes, it will work, but it is actually a pretty bad idea.

Let us assume that $y \in \{0, 1\}$ and $h_\theta \in [0, 1]$. Let us define $h_\theta(x)$ as follows:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

where $g(z) = \frac{1}{1+e^{-z}}$ is the *sigmoid (or logistic)* function.

Assuming we aims at learning $h_\theta(x)$ such that $\mathbb{P}(y = 1|x, \theta) = h_\theta(x)$ and $\mathbb{P}(y = 0|x, \theta) = 1 - h_\theta(x)$ we deduce that the likelihood is:

$$L(\theta) = \mathbb{P}(\vec{y}|x; \theta) = \prod_i \mathbb{P}(y^{(i)}|x^{(i)}, \theta) = \prod_i h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

As done before, it is easier to maximize the log of the likelihood $\ell(\theta) = \ln L(\theta)$, which leads to the same solution as the least square regression. However, it is not the same algorithm as the objective function is not the same.

There exists a faster method than the gradient algorithm: *Newton's method.*
The update rule is then:

$$\theta^{t+1} = \theta^t - \frac{f(\theta^t}{f'(\theta^t)}$$

Applying to the log-likelihood, we get:

$$\theta^{t+1} = \theta^t - H^{-1} \nabla_\theta \ell$$

Where $H$ is the Hessian matrix, $\nabla_\theta \ell$ is the gradient

**Advantages and disadvantages** The Newton's method is faster in term of number of steps, but at each iteration, we need to invert the $n \times n$ Hessian matrix, where $n$ is the number of features, which can be very costly.

## 4 Sparsity in Convex Optimisation for Supervised Machine Learning

**Reminder** With $S = \{z_i = (x_i, y_i)\}_{i=1}^m$ the set of trainings (i.i.d.) from an unknown joint distribution $\mathcal{D}_\mathcal{Z}$ over a space $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$

$$\min_h \sum_{i=1}^m \ell(y_i, h(x_i)) + \lambda ||h||$$

This can be rewritten as constrained problem:

$$\min_h \sum_{i=1}^m \ell(y_i, h(x_i)) \qquad \text{such that } ||h|| < c$$

We consider that the set $S$ is composed of $m$ examples in $\mathbb{R}^d$.

$$S = \begin{pmatrix} x_1 \\ . \\ . \\ . \\ x_i \\ . \\ . \\ . \\ x_m \end{pmatrix} = \begin{pmatrix} x^1 & ... & x^j & ... & x^d \end{pmatrix}$$

This set can be reduced:

- in columns $\rightarrow$ deletion of features, useful when $d$ is large compared to $m$
- in rows $\rightarrow$ deletion of examples (useful in $\ell_1$-SVM, CNN).
- in rank (e.g. PCA, LSA) $\rightarrow$ find the embedding space

Reducing can also:

- Prevent the algorithm from overfitting (curse of dimensionality) $\rightarrow$ Occam's razor principle
- Computational efficiency
- Interpretability: $\rightarrow$ understand the unerlying phenomenon

There are three categories of methods:

- *Filter approach* Use a criterion to filter, then learn after the treatment
- *Wrapper approach* Heuristic search of subset of variable, done with respect to the learning algorithm performance
- *Embedded approach* Use of sparsity-inducing norms :
  - Feature selection is part of the learning algorithm
  - All features processed during learning (ex: LASSO)

**Note** We prefer to use the $\ell_0$ norm rather than any other to induce sparsity. As usual, we will use the $\ell_1$-norm, as it is a good compromise between having a convex relation ($\leq 1$) and having a sparse solution ($\geq 1$).

### 4.1 $\ell_1$-norm Regularization

$\ell_1$-norm Regularization performs regularization as well as feature selection.

$\lambda$ can be seen as the range of value for what the value of the parameter is zero.

### 4.2 Optimization methods

We can use different tricks:

$$\mathbf{w}_j = \mathbf{w}_j^+ + \mathbf{w}_j^-$$

Or noticing that $||\mathbf{w}||_1 = \min_{\eta \geq 0} \frac{1}{2} \sum_{j=1}^d \dfrac{\mathbf{w}_j^2}{\eta_j} + \eta_j$

We can also use group lasso, if we have a partition of data to prioritise, and use the $\ell_1/\ell_2$-norm $= \sum_{g \in \mathcal{G}} ||\mathbf{w}_g||_2$ where $\mathcal{G}_k{}_{k=1}^K$ forms a partition of $\{1, ..., d\}$.

## 5 $k$-nearest neighbour

**Recall** The bayesian error is defined in definition 11.

**Bayesian Classifier** The Bayesian classifier predicts the optimal class $y^* \in \mathcal{Y}$ given an example $\mathbf{x} \in \mathcal{X}$ by applying the Maximum A Posteriori (MAP) decision rule:

$$\forall y_j \in \mathcal{Y}, p(y_i|\mathbf{x}) = \frac{p(\mathbf{x}|y_j).p(y_j)}{p(\mathbf{x})}$$
$$y^*(\mathbf{x} = \arg\max_c p(y_c|\mathbf{x})$$

that corresponds to $y^* = \arg\max_c p(\mathbf{x}|y_c).p(y_c)$, which is optimal from a probabilistic point of view.

However, this supposes to know $p(y_j)$ and $p(\mathbf{x}|y_j)$, which is not the case. We can estimate $p(y_j)$ by $\hat{p}(y_j) = \dfrac{|S_j|}{|S|}$, and $p(\mathbf{x}|y_j)$ by either a parametric method, that assume it follows a given statistical distribution (and we estimate it), or a non-parametric method, that estimates $p(\mathbf{x}|y_i)$ is locally estimated around $\mathbf{x}$.

An example of non-parametric method is $p(\mathbf{x}) \simeq \dfrac{k}{nV} = \hat{p}(\mathbf{x})$ (cf slides).

**Theorem 3.** *Let us denote by $\hat{p}_n(\mathbf{x}) = \frac{k_n}{nV_n}$ the estimate of $p(\mathbf{x})$ from a training sample $S$ of size $n$. When $n$ increases, $\hat{p}(\mathbf{x})$ converges to $p(\mathbf{x})$ if the following three conditions are fulfilled:*

- $\lim_{n\to\infty} V_n = 0$
- $\lim_{n\to\infty} k_n = \infty$
- $\lim_{n\to\infty} \dfrac{k_n}{n} = 0$

The $k$-Nearest Neighbour satisfy the previous theorem. It fixes a number $k_n$ of examples, and adapts a volume $V_n$ (e.g. an hypersphere centred at $\mathbf{x}$) such that $k_n$ examples are in the volume.

Using $k$-NN as a classifier, we get that $h(\mathbf{x}) = \arg\max_j = \dfrac{k_j}{k}$

**Propriety 2** (Convergence of the 1-nearest neighbour)**.** *Let $\mathbf{x}'$ be the nearest neighbour of $\mathbf{x}$, then*

$$\lim n \to \infty p(d(x, x') > \epsilon) =), \forall \epsilon 0$$

*and thus in $n \to \infty, p(y_j|\mathbf{x}') \simeq p(y_j|\mathbf{x})$.*

```
  Input: $x, S, d$
  Output: class of $x$
1 for $(\mathbf{x}', y') \in S$ do
2     Compute the distance $d(\mathbf{x}', \mathbf{x})$;
3 end
4 Sort the $n$ distances by increasing order;
5 Count the number of occurrences of each class $y_i$ among the $k$ nearest neighbours;
6 Assign to $\mathbf{x}$ the most frequent class ;
```

**Theorem 4.** *The generalization error $\epsilon_{1NN}$ of the 1-nearest neighbour rule is bounded by twice the (optimal) bayesian error $\epsilon^*$*

$$e_{1NN} \leq 2\epsilon^*$$

**Corollary 2.** *Half the informations about the true class of an example* $\mathbf{x}$ *is contained in its nearest neighbour* $\mathbf{x}'$

**Issue** This is only an asymptotic result → it needs a large number of training examples, and thus implies a large space/time complexity.

To cope with that, we can reduce the size of $S$ by keeping some examples only, and simplify the calculation of the nearest neighbour.

## 5.1 Data reduction techniques

It can be seen as a pretreatment on the set, that remove from $S$ the outliers and the examples of the bayesian error region.

```
  Input: $S$
  Output: $S_{cleaned}$
1 Split randomly $S$ into two subset $S_1$ and $S_2$;
2 while no stabilization of $S_1$ and $S_2$ do
3     Classify $S_1$ with $S_2$ using 1-NN rule;
4     Remove from $S_1$ the misclassified instances;
5     Classify $S_2$ with the new set $S_1$ using the $1 - NN$ rule;
6     Remove from $S_2$ the misclassified instances;
7 end
8 $S_{cleaned} = S_1 \cup S_2$;
```

## 5.2 Speeding-up the nearest-neighbour calculation

Most of the approach are based on the *triangle inequality property.*

# 6 Ensemble Methods

Many classifiers can be induced from the same task:

- Different algorithm
- Different hyperparameters
- Different training set, with different representations

```
Input: $S$
Output: STORAGE
STORAGE $\leftarrow \emptyset$;
DUSTBIN $\leftarrow \emptyset$;
Draw randomly a training example from $S$ and put it in STORAGE;
while no stabilization of STORAGE do
    for earch $\mathbf{x}_i \in S$ do
        if $\mathbf{x}_i$ is correctly classified with STORAGE using the 1-NN rule then
            DUSTBIN $\leftarrow \mathbf{x}_i$
        else
            STORAGE $\leftarrow \mathbf{x}_i$
        end
    end
end
```

**Model selection vs ensemble method** Rather than selecting the best model (w.r.t. some cross-validation procedure), why not try combining the whole set of classifier and taking adventaged of their diversity?

**Definition 16.** *Ensemble methods are learning algorithms that constructs a set of classifiers $h_1, ..., h_T$ whose individual decisions are combined in some way to classify new examples.*

For a classifier to be efficient, there must be:

- The individual classifier accurate, i.e. they have an error rate better than random guessing
- The classifiers are diverse, i.e. they make different errors on new data points.

**How combining different algorithms?** In most applications of machine learning, the true function $f$ cannot be represented by any of the hypotheses in $\mathcal{H}$.

By forming weighted sums of hypothesis drawn from $\mathcal{H}$, it may be possible to expand the space of representable functions.

**Computational problem:** Many learning algorithms works by performing some form of local search that may get suck in local optima. An ensemble constructed by running the local search from many different starting points may provide a better approximation to the unknown function.

There are two main categories of ensemble methods which depend on the origin of the diversity brought by the hypotheses.

**Heterogeneous ensemble method** Several classifiers are generated by applying different learning algorithm on the same dataset

**Homogeneous ensemble method** Several hypotheses are generated from a single learning algorithm on different dataset (i.e. modifying the statistical distribution of the training examples used to build $h_t$).

## 6.1 Heterogeneous ensemble method

### 6.1.1 Stacking

We learn $T$ hypotheses $h_1, ..., h_T$ with $T$ different learning algorithms. The decision (scores) of these hypothesis on $x$ is seen as new feature, and we learn a meta-hypothesis in this new $T$ dimensional space.

#### 6.1.2 Cascade generation

We learn a hypothesis $h_1$ with a learning algorithm $L_1$. We classify the learning example with $h_1$. Then we learn a hypothesis $h_2$ with a learning algorithm $L_2$ from the original features and the label (or score) predicted at the previous step. Finally, we classify the learning examples with $h_2$, and we repeat the process.

Drawback: we increase the dimensionality

### 6.2 Homogeneous ensemble method

#### 6.2.1 Bagging

```
// For ``bootstrap aggregating''
// TODO
```

#### 6.2.2 Random Forest

**Definition 17** (decision tree)**.** *A decision tree is a tree that can be learned by splitting $S$ into subsets based on a feature value test. This process is repeated on each derived subset in a recursive manner.*

**Definition 18.** *Decision Tree + Random Feature selection + Bagging = Random Forest*

We aim at generating diversity in decision trees. We build a model on successive resampling with replacement of $S$, and we make a majority vote to form the combined classifier. The decision are built with no pruning. While growing a tree, a random subset of $F$ features is selected from the $p$ original ones (typical values are $F = \lceil \log_2 p \rceil$ or $F = \lceil \sqrt{p} \rceil$).

## 7 Introduction to boosting

### 7.1 Theory of boosting

Bagging answer to the question: "How to choose the collection of examples that are the most useful", whereas boosting answer to "How to combine them into a accurate prediction rule"?

**Definition 19.** *Boosting is a general method for improving (under some constraints) the accuracy of any given learning algorithm.*

*Boosting combines* weak *hypotheses (i.e.just better than a random guessing) into a* strong *hypothesis (from a PAC theory point of view).*

```
1 Extract from $S$ a learning sample $S_1$, and use a learning algorithm $L$ to produce a first hypothesis $h_1$.
2 Generate a second learning sample $S_2$, in which an instance has roughly equal chance of being
    correctly or incorrectly classified by $h_1$. $L$ is used again to infer a new hypothesis $h_2$.
3 Generate a third learning sample $S_3$ by removing from $S$ the instance on which $h_1$ and $h_2$ disagree.
    Once again, $L$ is used to induce a third hypothesis $h_3$
4 The final hypothesis takes the majority vote of $h_1$, $h_2$ and $h_3$.
```

**Algorithm 4:** First boosting algorithm

## 8 Adaboost

Means "Adaptive-Boosting".

```
    Input: A learning sample $S$, a number of iterations $T$, a weak learner L
    Output: A global hypothesis $H_T$
 1  forall $i$ fron 1 to $m$ do
 2      $D_1(\mathbf{x}_i) = 1/m$;
 3  end
 4  forall $t$ from 1 to $T$ do
 5      $h_t = \mathrm{L}(S, \mathbf{D}_t)$;
 6      $\hat{\epsilon} = \sum_{\mathbf{x}_i \text{ s.t } y_i \neq h_t(\mathbf{x}_i} D_t(\mathbf{x}_i)$;
 7      $\alpha_t = \frac{1}{2} \ln \frac{1-\hat{\epsilon}_t}{\hat{\epsilon}_t}$;
 8      forall $i$ from 1 to $m$ do
 9          $D_{t+1}(\mathbf{x}_i) = D_t(\mathbf{x}_i) \exp(-\alpha_t y_i h_t(\mathbf{x}_i))/Z_t$;
            // $Z_t$ is a normalization coefficient
10      end
11  end
12  $f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$;
13  return $H_T$ such that $H_T(\mathbf{x}) = sign(f(\mathbf{x}))$
```

## 8.1 Theoretical results on the empirical risk

**Theorem 5.** *The upper bound on the empirical error of $H_T$ is*

$$\epsilon_{H_T} = \frac{1}{m} \sum_i [H(\mathbf{x}_i) \neq y_i] \leq \frac{1}{m} \sum_i \exp(-y_i f(\mathbf{x}_i)) = \prod_t Z_t$$

**Theorem 6.** *To minimize $Z_t$, the confidence coefficient $\alpha_t$ must be set to*

$$\alpha_t = \frac{1}{2} \ln \frac{1-\hat{\epsilon}_t}{\hat{\epsilon}_t}$$

**Theorem 7.** *Exponential decrease of the empirical risk*

$$\prod_t (Z_t) = \prod_t \sqrt{1 - 4\gamma_t^2} < \exp(-2 \sum_t \gamma_t^2)$$

This means that the empirical risk exponentially decreases towards 0 with the number $T$ of iterations.

*Proof.* In the slides. □

**Expected behaviour**

- $\hat{\epsilon}_{H_T}$ decreases towards 0
- $\epsilon_{H_T}$ first decreases; then $H_T$ becomes too complex $\rightarrow$ overfitting

**Observed behaviour**

- $\hat{\epsilon}_{H_T}$ decreases (eventually) towards 0
- $\epsilon_{H_T}$drops and continues to decrease even when $\hat{\epsilon}_T$ has reached 0!

**Definition 20.** *The margin of an example is defined by*

$$margin(\mathbf{x}) = \frac{yf(x)}{\sum_t \alpha_t} = \frac{y \sum_t \alpha_t h_t(\mathbf{x})}{\sum_t \alpha_t}$$

Larger margins on the training set translate into a tighter upper bound on the generalization error.

Despite the increase of its complexity, the final performing classifier is becoming easier to build because of the increase of the margins.

Experimentally, the margins increase during the course of adaboost.

**Theorem 8.** *Let $\mathcal{H}$ be a class of classifiers with VC dim $d_h$ (i.e. the capacity of $\mathcal{H}$). For any $\delta > 0$ and $\theta > 0$, with probability $1-\delta$, any classifier ensemble $H_T$ built from $m$ learning examples satisfies:*

$$\epsilon_{H_T} = \hat{Pr}(margin(\mathbf{x}) \leq 0) + \mathcal{O}\Big(\sqrt{\frac{d_h}{m}\frac{\log^2(m/d_h)}{\theta^2} + \log(1/\delta)}\Big)$$

This bound depends on:

- Constant parameters $m$, $d_h$, $\theta$ and $\delta$
- The distribution of the margins on the learning examples $\hat{Pr}$

**Theorem 9.** *$\hat{Pr}(margin(\mathbf{x}) \leq \theta)$ exponentially decrease towards 0 with $T$.*

**Theorem 10.** *Let $margin(\mathbf{x})$ be the margin of an example ($\gamma$ is here the gain compared to a random guessing):*

$$\hat{Pr}(margin(\mathbf{x}) \leq \theta) \leq \big(\sqrt{(1-2\gamma)^{1-\theta}(1-2\gamma)^{1+\theta}}\big)^T$$

*If $\theta < \gamma$, this bound decrease exponentially with $T$.*

**Advantages** Works well with Decision Stumps.

**Drawbacks** Adaboost fail with weak classifiers that are too strong (kNN, ID3). For kNN, you don't take advantage of the diversity of the data (as it is locally decided). Besides, Adaboost does not work with too weak classifier: there are underfitting and low margins.

Adaboost is especially susceptible to overfitting when there are outliers (exponential increase of their weights).

**An other point of view on Adaboost** We can see it as a double optimization procedure: optimizing the statistical distribution of the data ($D_t$) and a distribution $\alpha_t$ over the weak hypotheses.

**Definition 21.** *The* Edge *$E_t$ of a hypothesis $h_t$ for a given distribution $D_t$ on the $m$ training examples is*

$$E_t = \sum_{i=1}^{m} y_i h_t(x_i) \times D_t(x_i)$$

Adaboost aims at minimizing the maximum edge of past hypothesis while maximizing the minimum margin if the weak hypotheses.

# 9 SVM

## 9.1 Introduction

Found in the mid 90's ; very good performances, strong theoretical properties, efficient learning event in potentially infinite feature space.

## 9.2 Perceptron algorithm

```
Input: Training Set
k ← 0, w_k ← 0⃗, b_k ← 0, R ← max_{i=1}^n ||x_i||_2 repeat
    for i=1 to n do
        if y_i(⟨w_k, x_i⟩ + b_k) ≤ 0 then
            w_{k+1} ← w_k + η y_i x_i
            b_{k+1} ← b_k + η y_i R^2
            k ← k + 1
        end
    end
until no mistake is made;
return k, w_k, b_k
```

Dual form:

It's a way to learn a linear classifier. The dual representation of the problem allows us to learn the classifier in another representation space, where we only need inner products between training instances

The solution is not unique (we can solve it by taking the solution that maximizes the margins), and works only if the data is linearly separable.

## 9.3 Kernel trick

The principle of SVM is to project data into a more general space.
$\rightarrow$ we fall into the curse of dimensionality!

**Definition 22.** *A function $K : \mathcal{X} \times \mathcal{X} \rightarrow [-1; 1]$ is a (Mercel)* kernel *if there exists a function $\Phi : \mathcal{X} \rightarrow \mathcal{F}$ such that $K(x, x') = \langle \Phi(x), \Phi(x') \rangle$. Equivalently, we say that $K$ is a kernel if it is PSD*[1].

**Theorem 11** (Construction of kernels from other kernels)**.** *Let $\mathcal{X} \subseteq \mathbb{R}^d, \Phi : \mathcal{X} \rightarrow \mathbb{R}^m, f : \mathcal{X} \rightarrow \mathbb{R}, p \in \mathbb{N}$ and $a \in \mathbb{R}_+$. Let $K_1$ and $K_2$ be two kernels on $\mathcal{X} \times \mathcal{X}$, and $K_3$ be a kernel on $\mathbb{R}^m \times \mathbb{R}^m$. Let $B$ be a PSD matrix. Then the following functions are kernels:*

$$
\begin{aligned}
K(x, x') &= K_1(x, x') + K_2(x, x') \\
K(x, x') &= aK_1(x, x') \\
K(x, x') &= K_1(x, x')K_2(x, x') \\
K(x, x') &= f(x)f(x') \\
K(x, x') &= K_3(\Phi(x), \Phi(x')) \\
K(x, x') &= x^T B x'
\end{aligned}
$$

Using a kernel $K$ in the dual form of the Perceptron algorithm, we replace $\langle x_i, x_j \rangle$ by $\langle \Phi(x_i), \Phi(x_j) \rangle = K(x_i, x_j)$. Thus there is no need to represent $\Phi(x)$ explicitly, and the dimension of $\mathcal{F}$ does not affect the evaluation of $h$ nor the number of parameter to learn. In fact, we don't even need to know $\Phi$!

[1]Positive Symmetric Defined

## 9.4 Hard-Margin SVM

- Learn a linear classifier
- In the feature space induced by a kernel using the kernel trick
- Maximization of the margin of the hyperplane
- Done by convex optimization

**Definition 23** (SVM hard-margin primal form)**.** *Given a training set $T = \{z_i = (x_i, y_i)\}_{i=1}^n$ of a linearly-separable instances. The largest-margin hyperplane $(w^*, b^*)$ separating the instances of $T$ is the solution to the following optimization problem:*

$$\min_{w,b} \frac{1}{2}||w||_2^2$$

*subject to*

$$y_i(\langle w, x_i \rangle + b) > 1, 1 \leq i \leq n$$

**Remarks** We minimize $||W||_2^2$ to maximize the margin and keep the objective convex.

This can be solved efficiently as it is a strictly convex Quadratic Program (QP)