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 → 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 $\updownarrow : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}^+$ 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 ℓ 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)*

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

$\mathcal{R}^{\ell_{0/1}}$ 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 λ is a constant penalizing "too complex" models, and $||.||_p$ a ℓ_p-norm over the classifier h.

Definition 7 (ℓ_p-norm)**.** *If θ is a d-dimensional vector:*

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

The ℓ_2-norm is used to reduce the risk of overfitting (it decreases the large values of the model), and the ℓ_1 also allows the induction of sparse models - i.e. with less features (example: LASSO or ℓ_1-SVM).

Remark Increasing θ with the ℓ_1-norm causes more and more of the parameters θ_j to be driven to zero. The gradient on the ℓ_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 θ a theoretical parameter ($\mathcal{R}(h)$ in our case) and $\hat{\theta}$ an estimate of θ ($\hat{\mathcal{R}}(h)$ in our case). Let $B = \mathbb{E}(\theta) - \theta$ be the bias of $\hat{\theta}$ w.r.t. θ. The MSE assesses the quality of θ in terms of its variation and unbiasedness. It is the expected value of the square loss between $\hat{\theta}$ and θ.*

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

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 ϵ^* 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 $\hat{\mathrm{R}}(h)$ 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 ℓ 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 β 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 ℓ 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 σ comes from the σ admissibility of $\ell(h, z) = c(h(x), y)$ where c is the associated cost function.

Definition 13 (σ-admissibility)**.** *A loss function ℓ is σ-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 $\hat{\mathcal{R}}'(h)$
Split S randomly in k subsets $S_1, ..., S_k$ (if $k = m$, leave-one-out CV);
for *i=1 to k* **do**
  Run L on $S - S_i$ and induce classifier h_i;
end
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 θ 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 θ (e.g. $\vec{\theta} = 0$ or some randomly chosen vector.)
- Updateθ's values so that to reduce $J(\theta)$) (by computing partial derivatives of $J(\theta)$ w.r.t. θ).
- 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 θ 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.

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

Algorithm 2: Batch Gradient Descending Algorithm

Initialized $\vec{\theta}$;
repeat
  for *j=1 to m* **do**
    $\forall \theta_i$ of θ, $\theta_i := \theta_i - \alpha (h_\theta(x^{(j)}) - y^{(j)}) x_i^{(j)}$
  end
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 θ 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 α For some specific examples, $J(\theta)$ may increase. This may occur in some situations where θ is large ("zigzag" effect). To prevent the parameters θ'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 α 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, $\nabla_\theta J(\theta) \stackrel{\text{set}}{=} \vec{0}$ 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 ℓ_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 ℓ_0 norm rather than any other to induce sparsity. As usual, we will use the ℓ_1-norm, as it is a good compromise between having a convex relation (≤ 1) and having a sparse solution (≥ 1).

4.1 ℓ_1-norm Regularization

ℓ_1-norm Regularization performs regularization as well as feature selection.

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

We can also use group lasso, if we have a partition of data to prioritise, and use the ℓ_1/ℓ_2-norm $= \sum_{g \in \mathcal{G}} ||\mathbf{w}_g||_2$ where $\mathcal{G}_k{}_{k=1}^{K}$ forms a partition of $\{1, ..., d\}$.