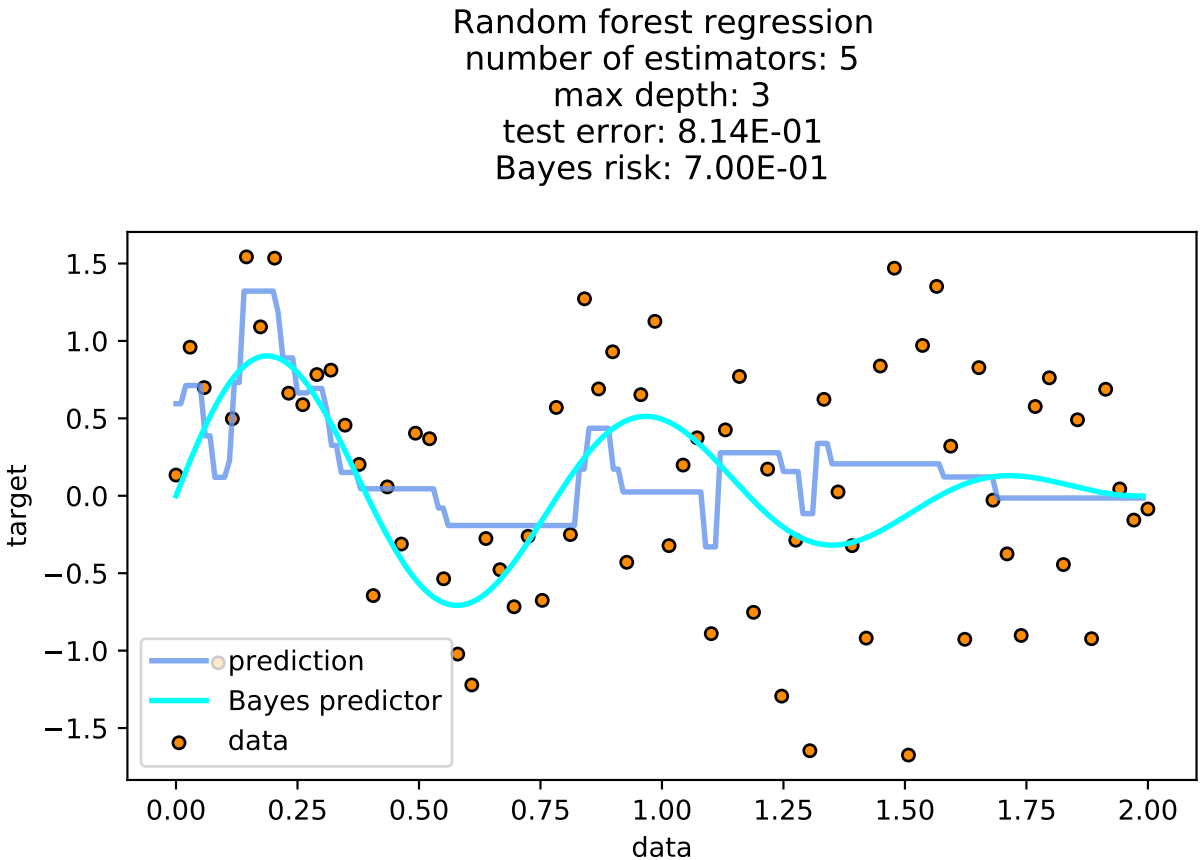
Fondamentaux théoriques du machine learning



- Risks and risk decompositions
- Probabilistic modelling
- Optimization of neural networks
  - Difficulties of optimizing neural networks
  - Specific methods for neural networks
- Classification and regression trees
  - Decision trees
  - Construction of a decision tree
  - Tree pruning
- Ensemble learning
  - Bagging
  - Random forest
  - Boosting

Risks and risk decompositions

Notion of risk decomposition

Context : usual supervised learning.

- empirical risk of estimator f : $R_n(f)$ (random variable)
- risk (real risk, generalization error) of estimator f : $R(f)$ (real number).
- dataset D_n.

We would like to interpret the risk of any estimator.

Convergence of empirical risk

We fix $f \in H$ (hypothesis space). We assume that the samples (x_i, y_i) are i.i.d, with the distribution of (X, Y), noted ρ. Then, according to the law of large numbers, under some assumptions (for instance, if the empirical risks are bounded), we have that in probability :

$$\lim_{n \to +\infty} R_n(f) = R(f) \tag{1}$$

The empirical risk of a fixed f converges to its real risk.

Notion of risk decomposition

We would like to compare $R(g)$, for some estimator g, to $R^* = R(f^*)$, f^* being the Bayes estimator.

First decomposition

- f^* : Bayes predictor
- F : Hypothesis space
- g : some predictor ($\in F$).

$$R(g) - R^* = \Big(R(g) - \inf_{f \in F} R(f)\Big) + \Big(\inf_{f \in F} R(f) - R^*\Big) \qquad (2)$$

First decomposition

- f^* : Bayes predictor
- F : Hypothesis space
- g : some predictor
- f_n : empirical risk minimizer

Most of the time, we will be interested in such a decomposition for the learned estimator, e.g. the empirical risk minimizer f_n. Now, $R(f_n)$ is a **random variable** !

$$E[R(f_n)] - R^* = \Big(E[R(f_n)] - \inf_{f \in F} R(f)\Big) + \Big(\inf_{f \in F} R(f) - R^*\Big) \quad (3)$$

Underfitting and overfitting

Approximation error (bias) : depends on f^* and F, not on f_n, D_n.

$$\inf_{f \in F} R(f) - R^* \geq 0$$

Estimation error (variance, fluctuation error, stochastic error) : depends on D_n, F, f_n.

$$E(R(f_n)) - \inf_{f \in F} R(f) \geq 0$$

- too small F (compared to f^*) : underfitting (large bias, small variance)
- too large F (compared to n) : overffitting (small bias, large variance)

Deterministic bound on the estimation error

We consider the best estimator in the hypothesis space F.

$$f_a = \arg\min_{h \in F} R(h)$$

Exercice 1 : Show that

$$R(f_n) - R(f_a) \leq 2 \sup_{h \in F} |R(h) - R_n(h)| \tag{4}$$

Deterministic bound on the estimation error

$$R(f_n) - R(f_a) \leq 2 \sup_{h \in F} |R(h) - R_n(h)| \tag{5}$$

Later in the course, based on **concentration inequalities** we will further build on this result and prove a probabilistic bound of the form

$$R(f_n) - R(f_a) \leq \frac{C}{\sqrt{n}} \tag{6}$$

(remember that by definition $0 \leq R(f_n) - R(f_a)$)

Approximate solution

- In machine learning, it is often not necessary to find the actual minimizer of the empirical risk , as there is an estimation error of $\mathcal{O}(\frac{1}{\sqrt{n}})$. [Bottou and Bousquet, 2009,]
- We can use an approximate solution $\hat{f}_n$, such that

$$R_n(\hat{f}_n) \leq R_n(f_n) + \rho \tag{7}$$

  with ρ a predefined tolerance.
- This important because in large-scale ML, the **computation time** needs to be optimized.

Approximate solution

This gives a new risk decomposition :

$$\begin{aligned} E[R(\hat{f}_n)] - R^* = & \left(E[R(\hat{f}_n)] - E[R(f_n)] \right) \\ & + \left(E[R(f_n)] - \inf_{f \in F} R(f) \right) \\ & + \left(\inf_{f \in F} R(f) - R^* \right) \end{aligned} \tag{8}$$

$$
\begin{aligned}
E[R(\hat{f}_n)] - R^* = & \left(E[R(\hat{f}_n)] - E[R(f_n)] \right) \\
& + \left(E[R(f_n)] - \inf_{f \in F} R(f) \right) \\
& + \left(\inf_{f \in F} R(f) - R^* \right)
\end{aligned} \tag{9}
$$

$E[R(\hat{f}_n)] - E[R(\tilde{f}_n)]$ is the **optimization error.**
To conclude, we have :

- an approximation error
- an estimation error
- an optimization error

Context

We are given a set of observations $\{y_1, \ldots, y_n\} \in \mathcal{Y}$ that we assume are generated i.i.d from an unknown distribution. We look for a **probabilistic model** that explains well the data. We could for instance use this model to generate new data, that would be statistically similar to the observed ones.

Density estimation

We will consider **parametric models** for density estimation.

Definition

Parametric model

Let $\Theta \subset \mathbb{R}^p$. A parametric model $\mathcal{P}$ is a set of probability distributions on $\mathcal{Y}$, indexed by elements of Θ.

$$\mathcal{P} = \{p_\theta | \theta \in \Theta\}$$

Examples :

- Bernoulli model (parameter p)
- Gaussian model (parameter (μ, σ))
- Binomial model (parameter (n, p))

Objective

If we assume that the data were generated from some $p_{\theta^*} \in \mathcal{P}$, with a unknown parameter θ^*, our goal if to find a good estimation of θ. If the data are indeed generated by a distribution in $\mathcal{P}$, the problem is said to be **well specified**. Otherwise, the problem is said to be **misspecified**.

Definition

Likelihood

Let $\mathcal{P} = \{p_\theta, \theta \in \Theta\}$ be a parametric model.
Given $y \in \mathcal{Y}$, the **likelihood** is the function :

$$\theta \mapsto L(\theta|y) = p_\theta(y) \tag{10}$$

Given $D_n = (y_1, \ldots, y_n)$, the likelihood $L(.|D_n)$ is the function :

$$\theta \mapsto L(\theta|D_n) = \prod_{i=1}^{n} p_\theta(y_i) \tag{11}$$

The **maximum likelihood estimator** (MLE) is the parameter θ that maximises the likelihood :

$$\hat{\theta}_n \in \arg\max_{\theta \in \Theta}(L(\theta|D_n)) \tag{12}$$

Remarks

- Since the samples y_i are assumed to be independent, the likelihood corresponds to the probability (or probability density) of observing the dataset according to p_θ.
- We often maximise the log of the likelihood, as it is easier to differentiate a sum. Since log is an increasing function, the MLE is also the maximiser of the log of L.

Example 1

Exercice 2 : We observe the data $(1, 0)$. We model these data with a Bernoulli distribution of parameter p.

- What is the likelihood of these observations as a function of p ?
- What is the value $\hat{p}$ that maximizes this likelihood ?

Example 2

Exercice 3 : We observe the data $(1, 0, 1)$ (same hypotheses)

- What is the likelihood of these observations as a function of p ?
- What is the value $\hat{p}$ that maximizes this likelihood ?

Example 3

We observe the data $(2.5, 3.5)$. We assume that these data come from a normal law of parameters μ and σ.

$$\begin{aligned} L &= p(2.5|\mu,\sigma)p(3.5|\mu,\sigma) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{2.5-\mu}{\sigma})^2} \times \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{3.5-\mu}{\sigma})^2} \end{aligned} \tag{13}$$

We wan show that the likelihood is maximum for :

- $\hat{\mu} = \frac{2.5+3.5}{2}$
- $\hat{\sigma^2} = \frac{(2.5-\hat{\mu})^2+(3.5-\hat{\mu})^2}{2}$

ERM

In the context of density estimation, we can define a loss function as the **negative log-likelihood.**

$$\Theta \times \mathcal{Y} \mapsto -\log(p_\theta(y))$$

Given this loss, the risk writes :

$$R(\theta) = E_Y[-\log\left(p_\theta(y)\right)]$$

and the empirical risk :

$$R_n(\theta) = -\frac{1}{n}\sum_{i=1}^{n} \log\left(p_\theta(y_i)\right)]$$

The MLE is then also the empirical risk minimizer.

KL divergence

The Kullback-Leibler divergence is a quantity used to compare two probability distributions.

Definition

Kullback-Leibler divergence
Given two distributions p and q, the KL divergence from p to q is defined as :

$$KL(p||q) = E_{Y\sim p}\left[\log \frac{p(Y)}{q(Y)}\right]$$

Lemma

If the data are generated by p_{θ^*}, then $KL(p_{\theta^*}||p_\theta)$ is the excess risk of p_θ, with the legative log-likelihood loss.

Link with supervised learning methods

Probabilistic modelling can provide an interesting interpretation of several supervised learning methods, such as :

- logistic regression
- ordinary least squares

In a supervised learning context, we replace the likelihood $p_\theta(y)$ by a **conditional** likelihood $p_\theta(y|x)$ (conditional modelling).

$$R_n(\theta) = -\frac{1}{n}\sum_{i=1}^{n} \log\left(p_\theta(y_i|x_i)\right) \tag{14}$$

Link with logistic regression

We consider a binary classification problem, with $\mathcal{Y} = \{0, 1\}$.
Let us now consider the probabilistic model such that

$$p_\theta(1|x) = \sigma(\theta^T x)$$

Equivalently, this model can be written (remember that $y = 0$ or $y = 1$)

$$p_\theta(y|x) = \big(\sigma(\theta^T x)\big)^y \big(1 - \sigma(\theta^T x)\big)^{1-y} \tag{15}$$

Exercice 4 : Show that the parameter θ with maximum likelihood is the logistic regression estimator θ_{logit} (cross entropy version).

Neural networks

References / tools :

- https://www.deeplearningbook.org/
- https://d2l.ai/
- https://mlelarge.github.io/dataflowr-web/dldiy_ens.html
- https://playground.tensorflow.org/
- http://www.jzliu.net/blog/simple-python-library-visualize-neural-network/

Learning representations / features

- $\mathcal{X} = \mathbb{R}^d$.
- $\mathcal{Y} = \mathbb{R}$.

A neural network with scalar output learns a feature map $\phi : \mathbb{R}^d \to \mathbb{R}^m$ **and** a linear regressor or classifier, $\theta \in \mathbb{R}^m$.

$$\forall x, f(x) = \langle \theta, \phi(x) \rangle \tag{16}$$

We can add a intercept (bias) by adding a dimension to θ and adding a component with a 1 to each $\phi(x)$.

Learning representations / features

- $\mathcal{X} = \mathbb{R}^d$.
- $\mathcal{Y} = \mathbb{R}$.

A neural network learns a feature map $\phi : \mathbb{R}^d \to \mathbb{R}^m$ **and** a linear regressor or classifier, $\theta \in \mathbb{R}^m$.

$$\forall x, f(x) = \langle \theta, \phi(x) \rangle \tag{17}$$

We can add a bias by adding a dimension to θ and adding a component with a 1 to each $\phi(x)$.
Remark : kernel methods use hardcoded features ϕ, that can be **implicit** (kernel trick).

Multi output

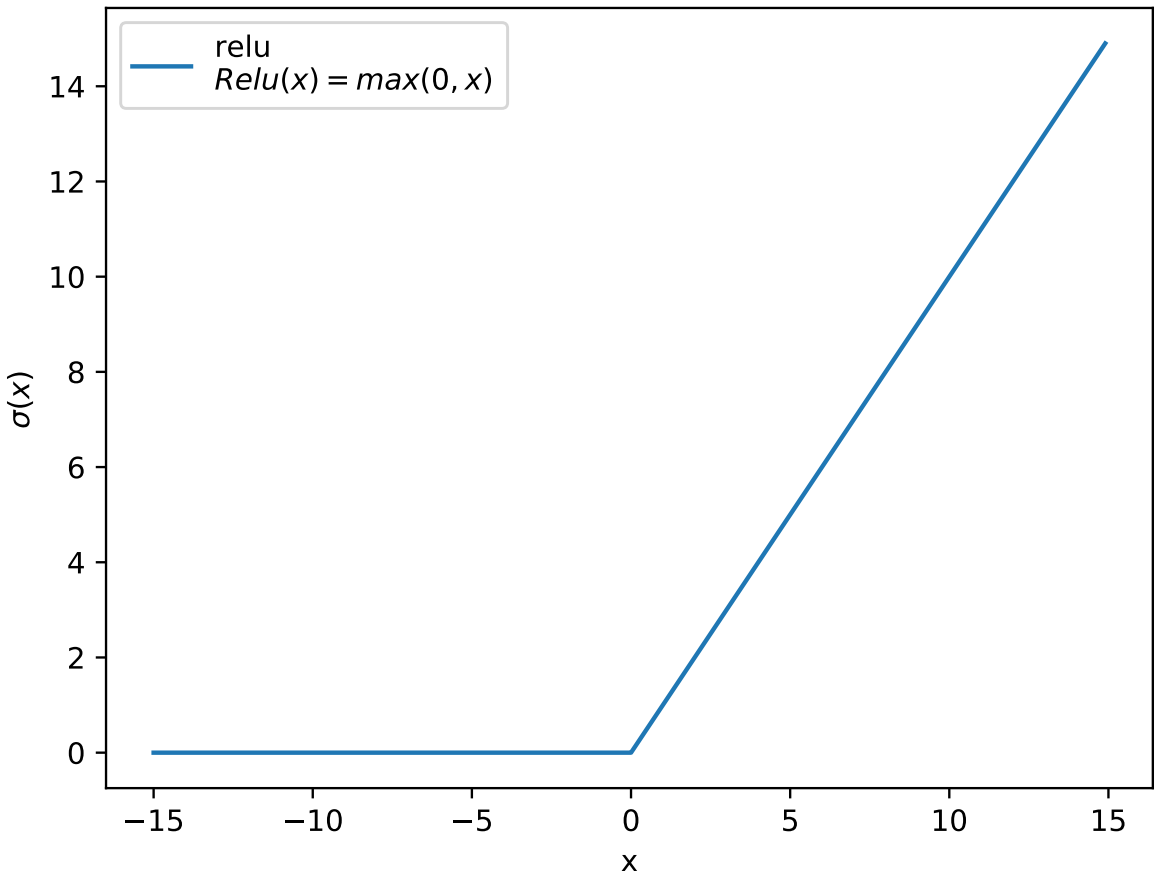
In order to learn a multidimensional output of dimension p ($\mathcal{Y} = \mathbb{R}^p$) , it is sufficient to lean a matrix $\theta \in \mathbb{R}^{m,p}$.

Single layer neural network

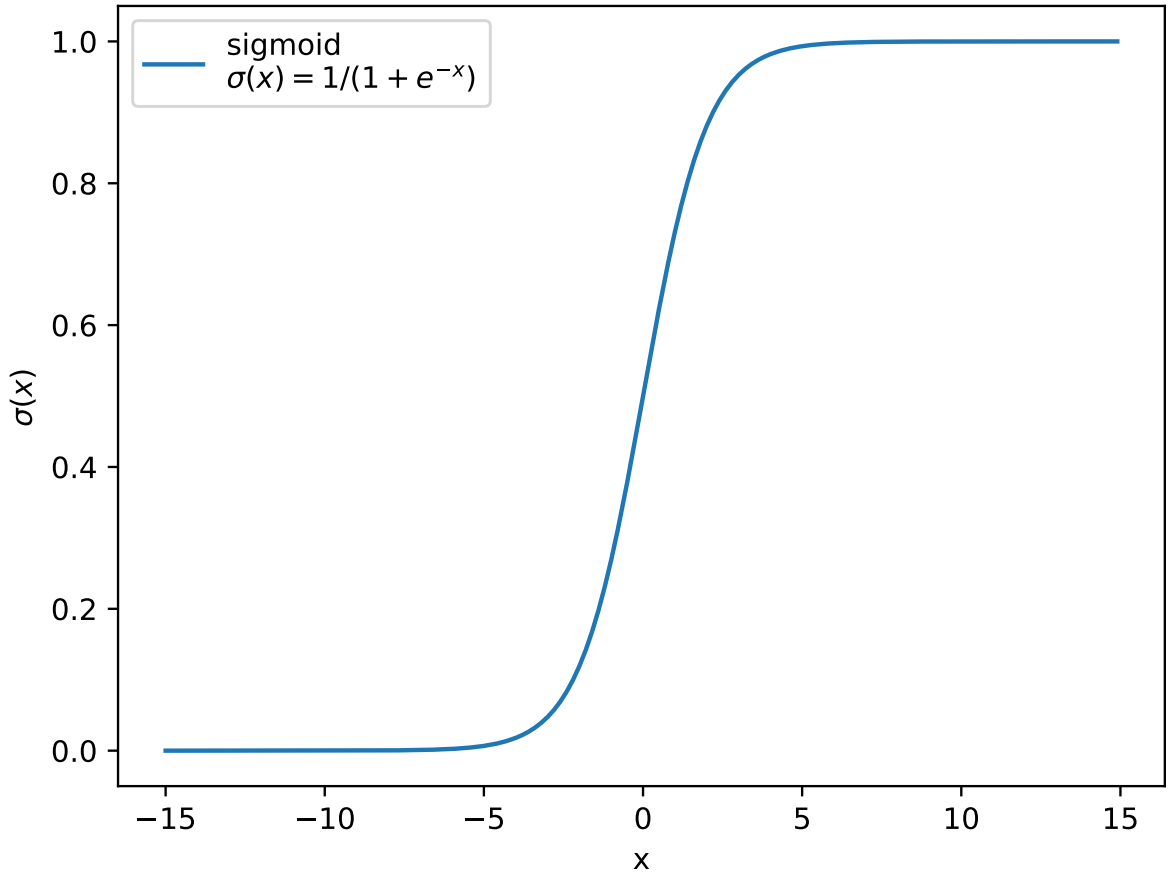
$$f(x) = \sum_{j=1}^{m} \theta_j \sigma(w_j^T x + b_j) \tag{18}$$

σ is an activation function (sigmoid, tanh, ReLu, etc).

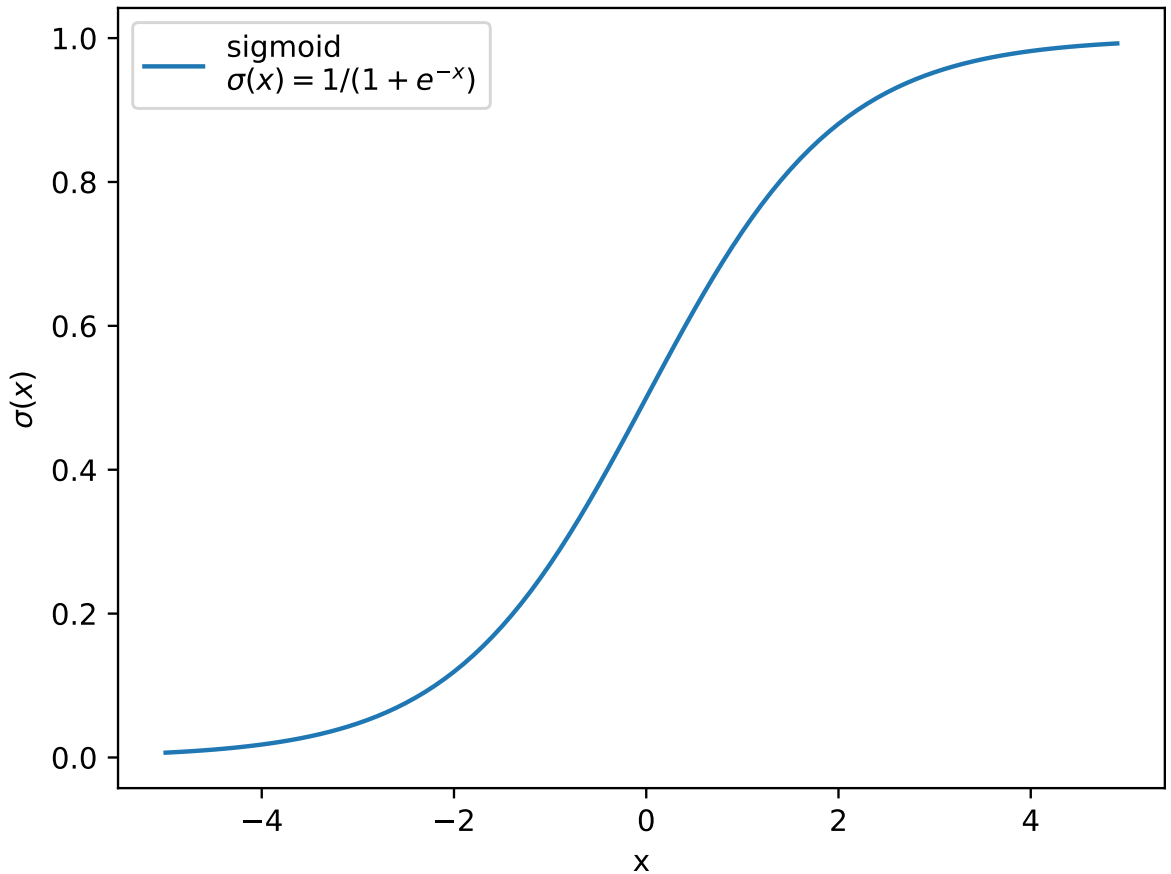
ReLU



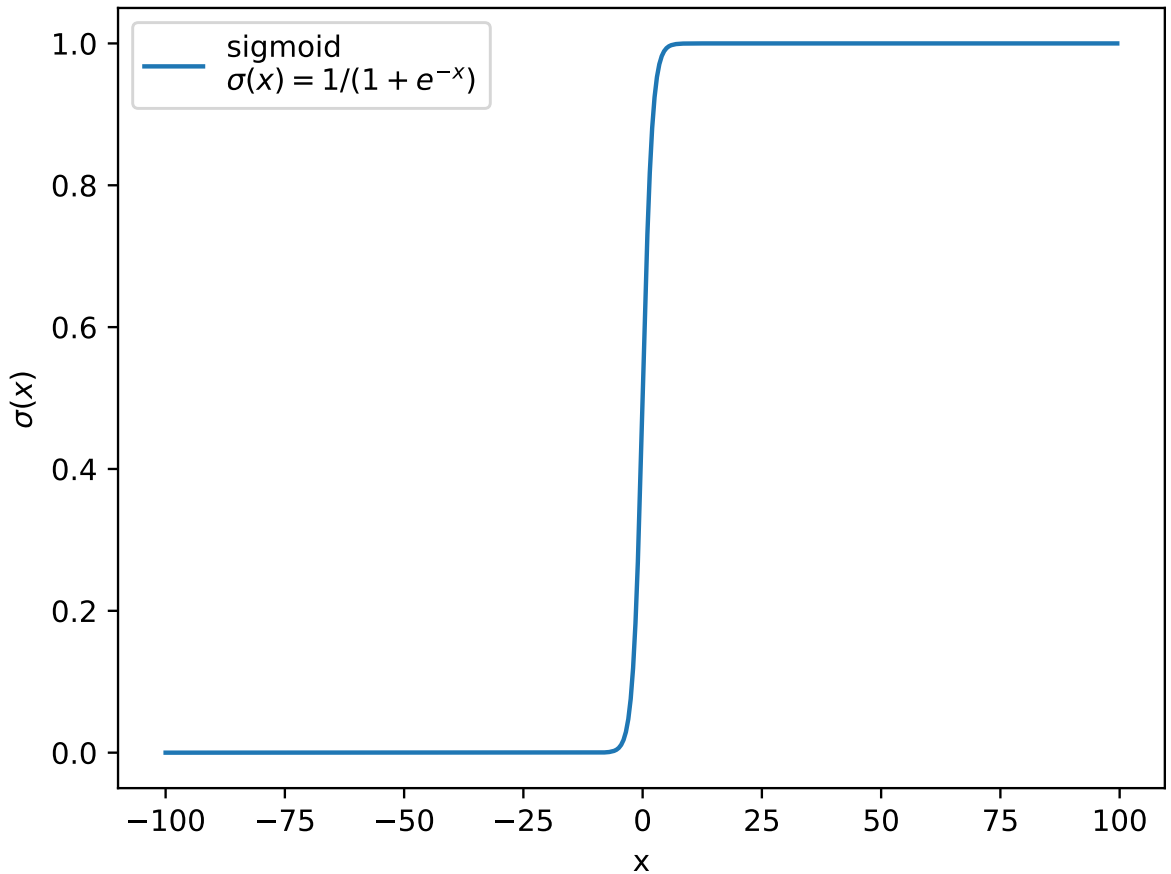
Sigmoid



Sigmoid



Sigmoid


sigmoid
$\sigma(x) = 1/(1 + e^{-x})$
1.0
0.8
0.6
0.4
0.2
0.0
$\sigma(x)$
−100
−75
−50
−25
0
25
50
75
100
x

Automatic differentiation

When working with neural networks, most used libraries implement automatic differentiation.

- tensorflow
- pytorch (autograd)

Optimizing neural networks

Optimizing neural networks comes with specific difficulties.

- the problem is non-convex
- there is often a large number of parameters (optimization in a high dimensional space)
- specific problems due to depth (vanishing gradients, see below)

`http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf`

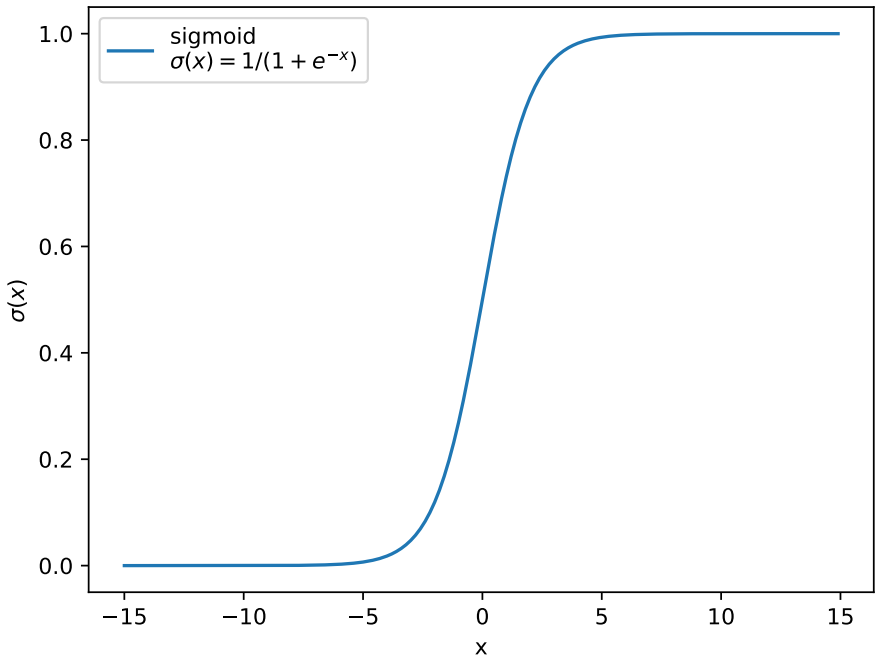
Non-convexity

We know that

- If f is increasing and convex and g is convex, then $f \circ g$ is convex.
- Is f in convex and g is linear, then $f \circ g$ is convex.

With neural networks, we are in neither of these cases, as the activations σ are non linear.
Hence **the objective function is non-convex**, and it remains difficult to understand why gradient based methods often perform well in practice

Vanishing gradients



Exercice 5 :

What is the maximum value of $|\sigma'(z)|$?

Exponentially decreasing gradients

- At each layer, the gradients are multiplied by a term of the form $\sigma'(u)$. Using a large number of layers leads to gradient norms that decrease rapidly when we move away from the output layer.
- This slows training down and caused deep learning to plateau for some years.
- Several initializations were necessary in order to obtain convergence, the result was unstable.

ReLU

The usage of ReLU solved this problem.
Other activation functions :
https://dashee87.github.io/deep%20learning/visualising-activation-functions-in-neural-networks/

SGD variants for neural networks

Several specific variations of SGD are commonly used for deep learning.
`https://pytorch.org/docs/stable/optim.html`

Specific methods for deep learning

Architectures :

- Convotutional networks
- Residual neural network (ResNet)

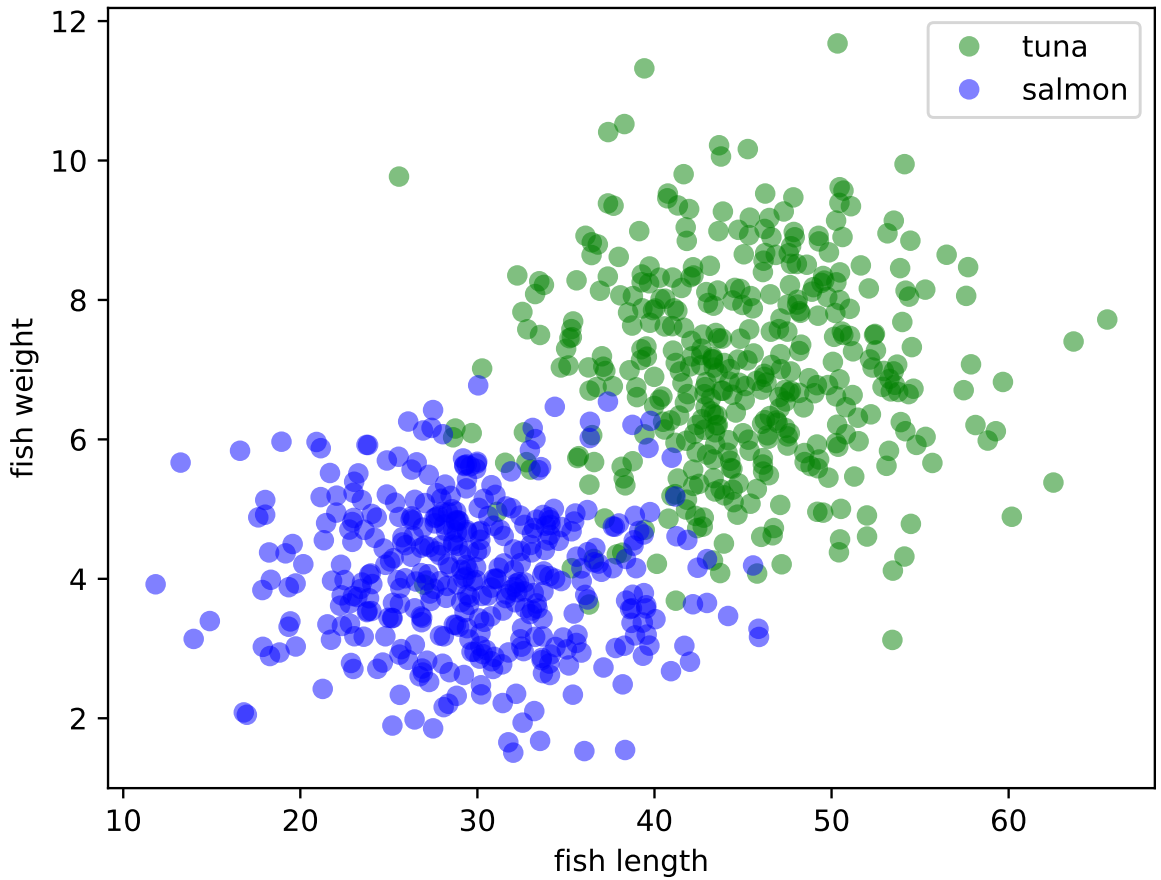
Optimization / regularization :

- dropout
- batch normalisation

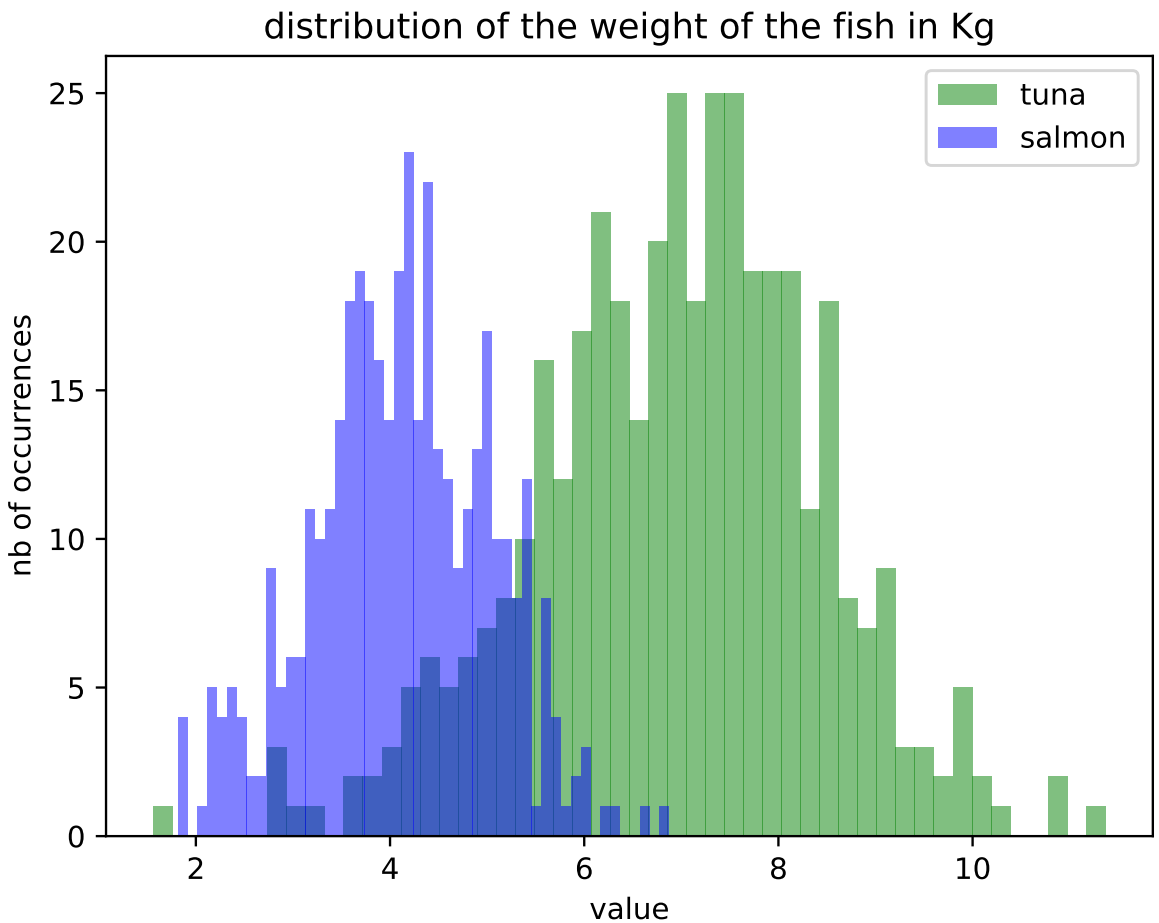
CART

- Segmentation method (partitionnement récursif), **binary splits.**
- First algorithm : **CART** (classification and regression tree, Breiman, 1984) [Breiman et al., 2017]

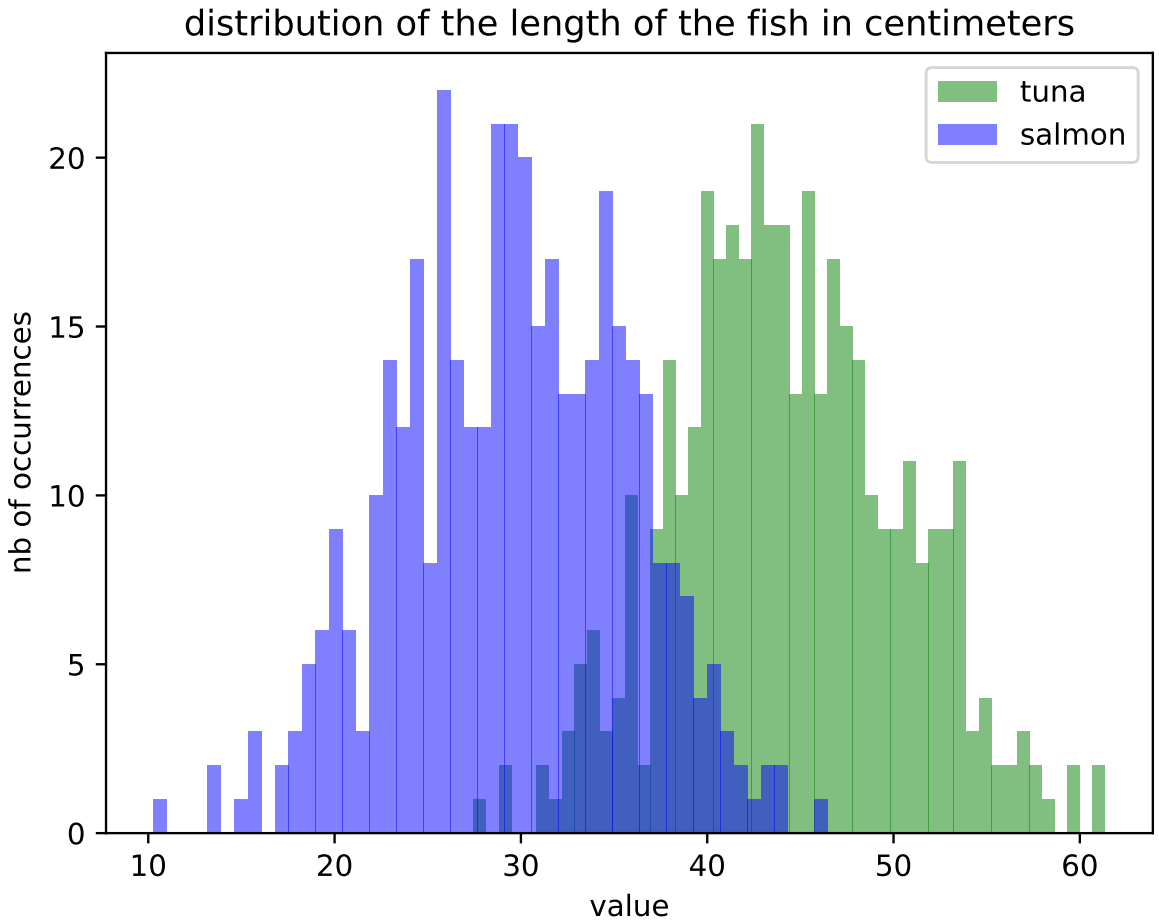
Example dataset : fishes



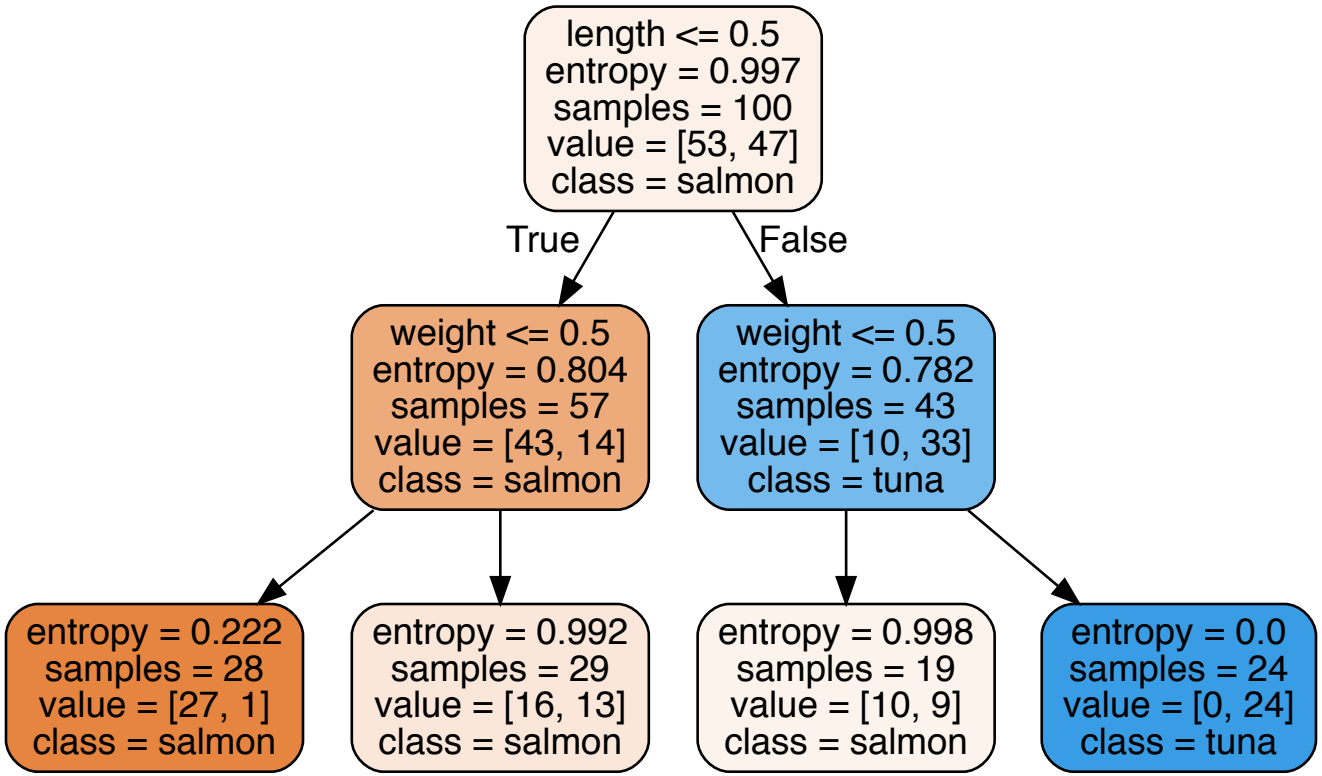
Fish dataset



Fish dataset

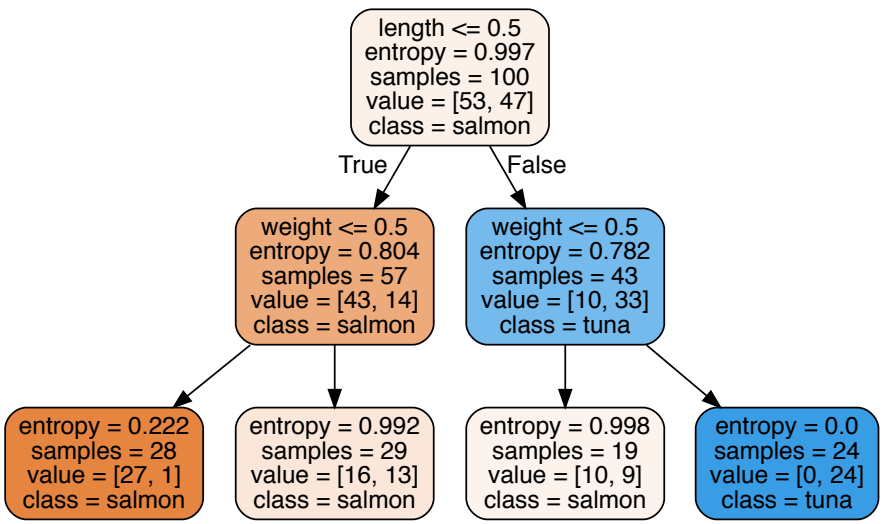


Example tree



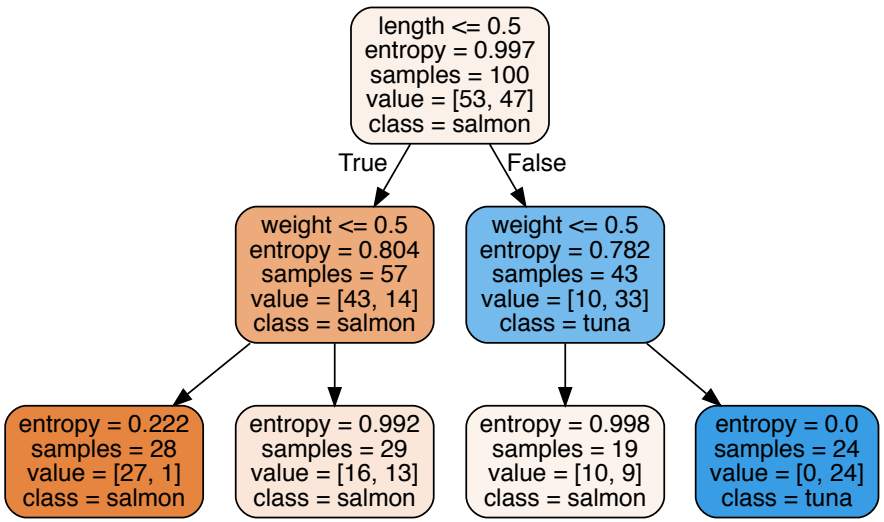
Splitting nodes

Each split should lead to more **homogeneous** nodes (in a sense that is to be formally defined)

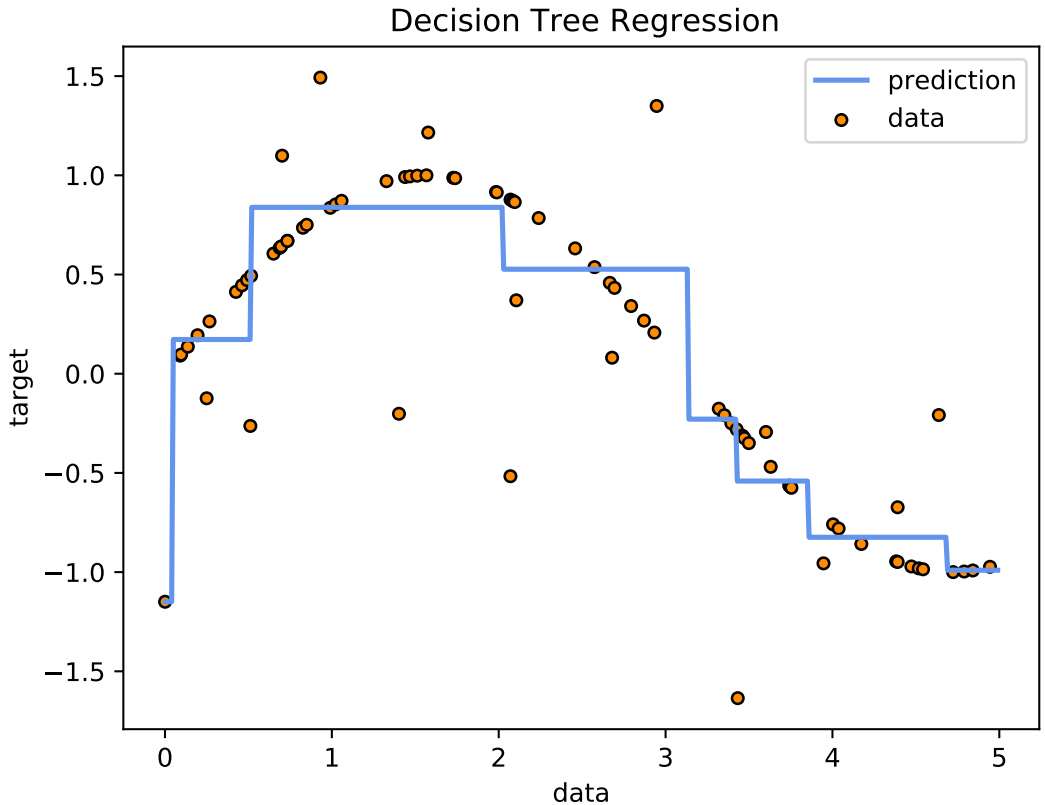


Splitting nodes

A leaf node corresponds to a predicted value.



Decision trees can be used for regression. In that case, the prediction is **piecewise constant**. It can be seen as a form of local averaging.



Construction of a tree

A split node corresponds to :

- a segmentation variable V
- a segmentation value / threshold if V is quantitative. If V is a class, it is a two-fold partitionning of the set of classes.

The construction of a tree requires a criterion to :

- choose the segmentation variable and value
- decide when a node is terminal (leaf node)
- associate a prediction value to all leaf nodes

Homogeneity

We want the homogeneity in the child nodes to be greater than in the parent node. Equivalently, we can define a non-negative heterogeneity function H that describes each node and that must be :

- 0 if the node is homogeneous (only one class or one output value is represented in this node)
- maximal if the values of y are uniformly distributed within the node.
- if $H(L)$ it the value of H for the left child node (and $H(R)$ for the right one), then the segmentation should minimize

$$H(L) + H(R) \tag{19}$$

Homogeneity criterion for regression

In the case of regression, $\mathcal{Y} = \mathbb{R}$ and the heterogeneity $H(n)$ of node n is its empirical variance :

$$H(n) = \frac{1}{|n|} \sum_{i \in n} (y_i - \bar{y}_n)^2 \tag{20}$$

Homogeneity criterion for classification

In the case of classification, $\mathcal{Y} = [1, \ldots, L]$ and several criteria exist.

Homogeneity criterion for classification : entropy (information gain)

- We use the convention that $0 \log(0) = 0$
- p_n^l is the proportion of class l in node n.

The **entropy** writes :

$$H(n) = -\sum_{l=1}^{L} p_n^l \log(p_n^l) \tag{21}$$

Exercice 6 : What are the maximum and minimum values of the entropy ?

Homogeneity criterion for classification : Gini impurity

The Gini impurity writes :

$$H(n) = \sum_{l=1}^{L} p_n^l (1 - p_n^l) \tag{22}$$

Homogeneity criterion for classification : Gini impurity

The Gini impurity writes :

$$H(n) = \sum_{l=1}^{L} p_n^l (1 - p_n^l) \tag{23}$$

Exercice 7 : Interpret the meaning of the Gini impurity in terms of probabilities, assuming that we predict according to the proportions p_n^l.

Homogeneity criterion for classification : misclassification probability

$$H(n) = 1 - \max_{l}(p_n^l) \tag{24}$$

If we predict the most represented class in n, then this is the misclassification probability.

Prediction for a leaf node

- **regression** : predict the average value in the node
- **classification** : several possibilities
  - most represented class in the node
  - most probable class *a posteriori* if the *a priori* probabilities are known (Bayesian probabilities)
  - class that costs the less in case of missclassification

Pruning

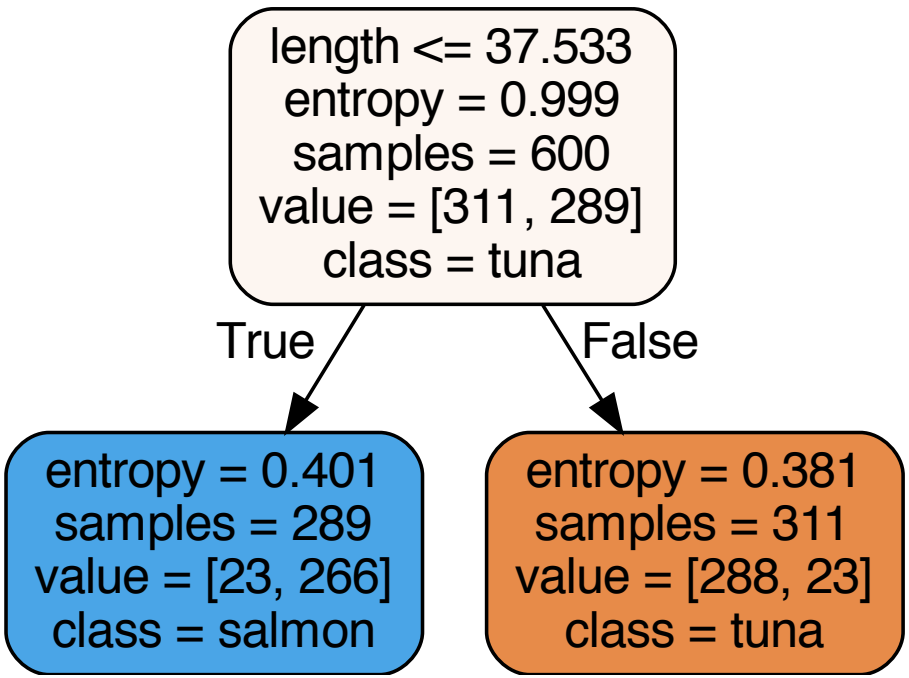
- If the segmentation is not restricted, the tree can fit the training dataset perfectly (overfitting).
- learning would then be highly dependent on the choice of the training dataset, leading to **instability.**
- To avoid it, **pruning** (élaguage) is used : it consists in restricting the size of the tree (and can be seen as an equivalent to regularization like in ridge regression or logistic regression).

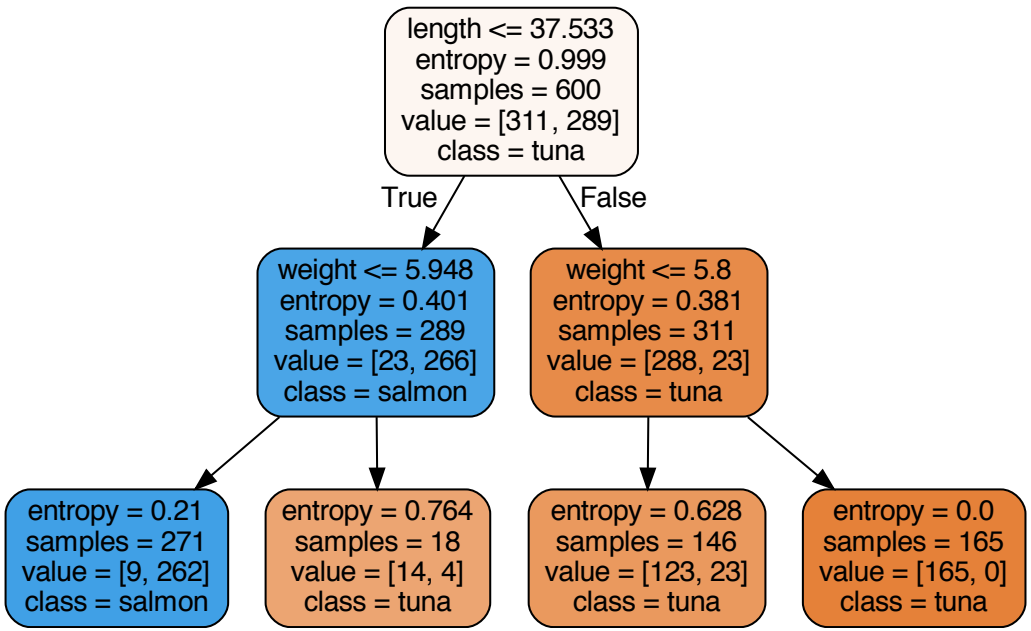
Pre-pruning by restricting divisions

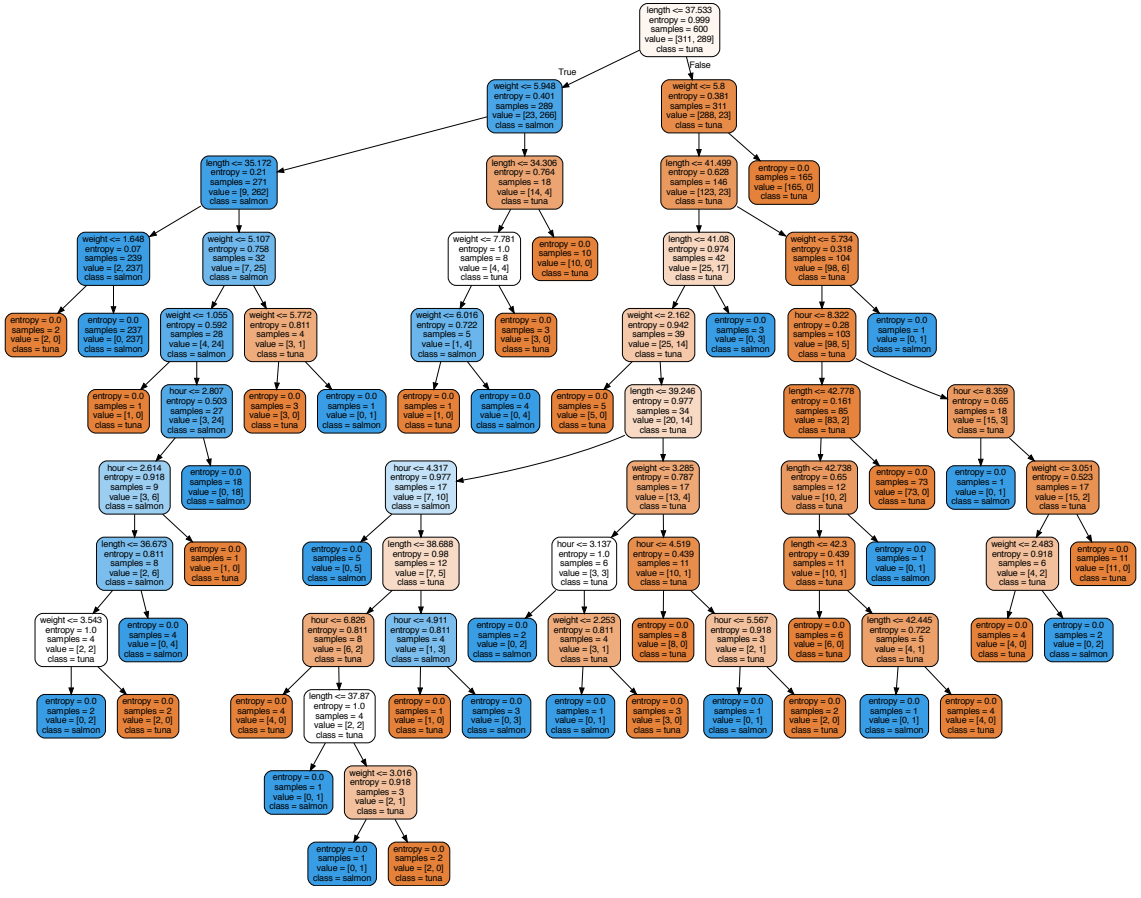
We can impose

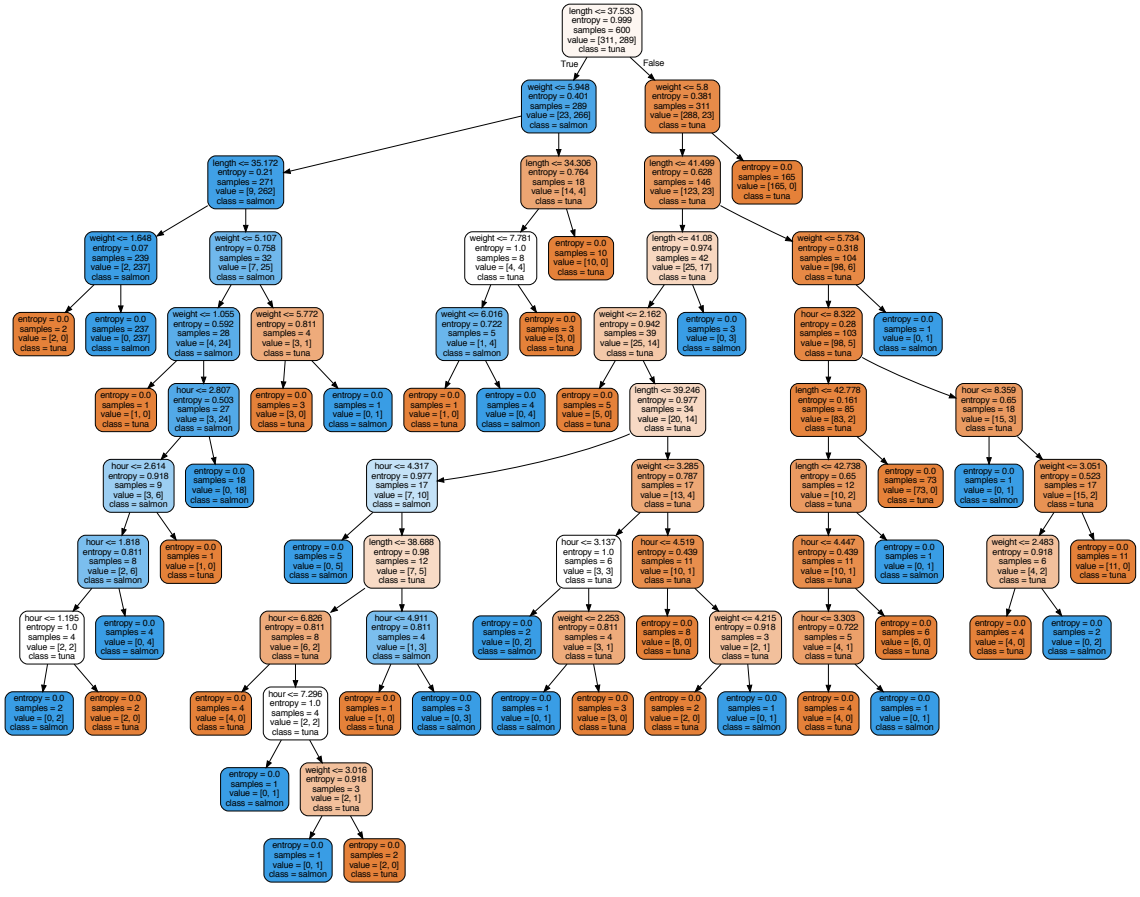
- minimum impurity decrease
- a minimum number of samples in a leaf node
- a minimum number of samples to authorize splitting a node

Simulation : fish dataset, max depth=1

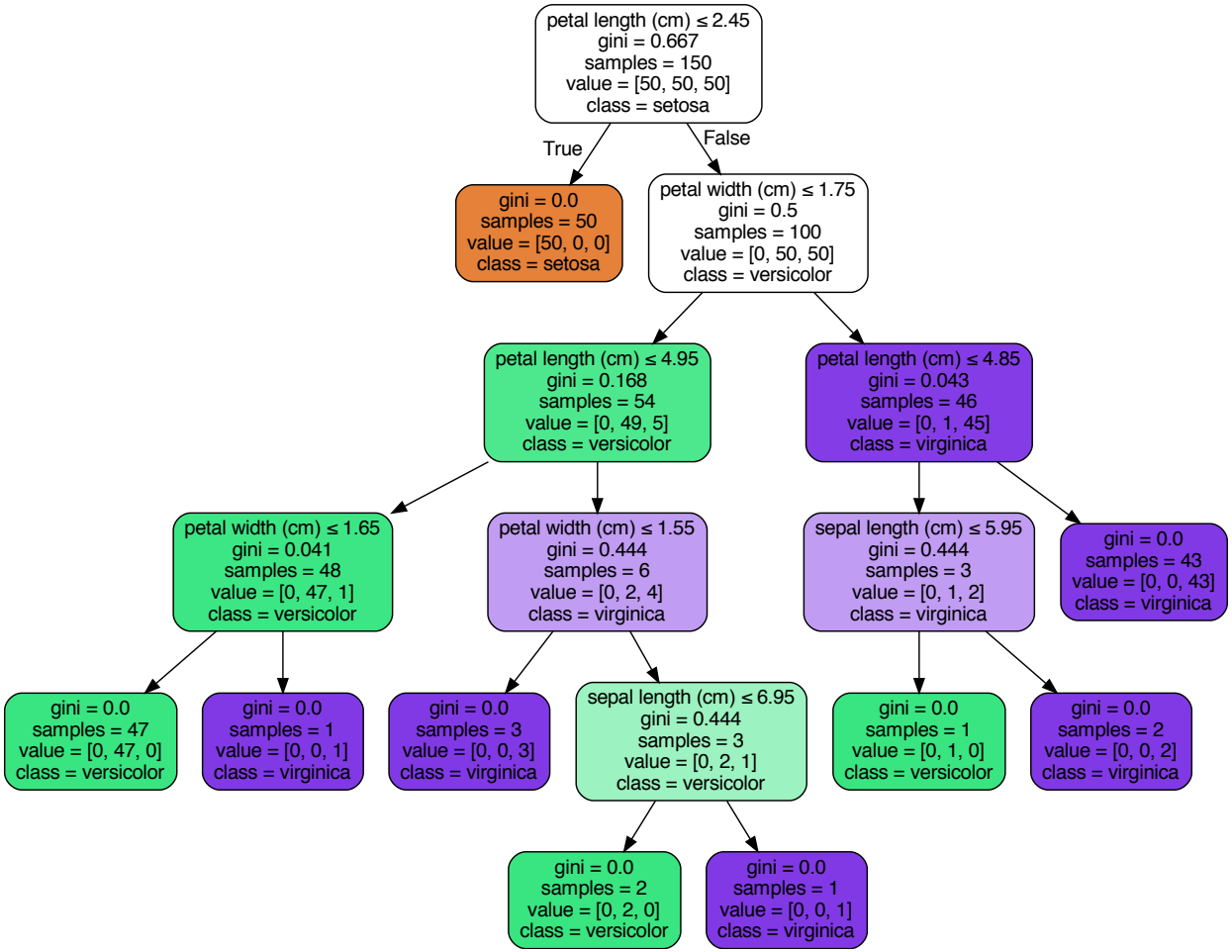



length <= 37.533
entropy = 0.999
samples = 600
value = [311, 289]
class = tuna
True
False
weight <= 5.948
entropy = 0.401
samples = 289
value = [23, 266]
class = salmon
weight <= 5.8
entropy = 0.381
samples = 311
value = [288, 23]
class = tuna
entropy = 0.21
samples = 271
value = [9, 262]
class = salmon
entropy = 0.764
samples = 18
value = [14, 4]
class = tuna
entropy = 0.628
samples = 146
value = [123, 23]
class = tuna
entropy = 0.0
samples = 165
value = [165, 0]
class = tuna


length <= 37.533
entropy = 0.999
samples = 600
value = [311, 289]
class = tuna
True
False
weight <= 5.948
entropy = 0.401
samples = 289
value = [23, 266]
class = salmon
weight <= 5.8
entropy = 0.381
samples = 311
value = [288, 23]
class = tuna
length <= 35.172
entropy = 0.21
samples = 271
value = [9, 262]
class = salmon
length <= 34.306
entropy = 0.764
samples = 18
value = [14, 4]
class = tuna
length <= 41.496
entropy = 0.628
samples = 146
value = [123, 23]
class = tuna
entropy = 0.0
samples = 165
value = [165, 0]
class = tuna
weight <= 1.648
entropy = 0.07
samples = 239
value = [2, 237]
class = salmon
weight <= 5.107
entropy = 0.758
samples = 32
value = [7, 25]
class = salmon
weight <= 7.781
entropy = 1.0
samples = 8
value = [4, 4]
class = tuna
entropy = 0.0
samples = 10
value = [10, 0]
class = tuna
length <= 41.08
entropy = 0.974
samples = 42
value = [25, 17]
class = tuna
weight <= 5.734
entropy = 0.318
samples = 104
value = [98, 6]
class = tuna
entropy = 0.0
samples = 2
value = [2, 0]
class = tuna
entropy = 0.0
samples = 237
value = [0, 237]
class = salmon
weight <= 1.055
entropy = 0.592
samples = 28
value = [4, 24]
class = salmon
weight <= 5.772
entropy = 0.811
samples = 4
value = [3, 1]
class = tuna
weight <= 6.016
entropy = 0.722
samples = 5
value = [1, 4]
class = salmon
entropy = 0.0
samples = 3
value = [3, 0]
class = tuna
weight <= 2.162
entropy = 0.942
samples = 39
value = [25, 14]
class = tuna
entropy = 0.0
samples = 3
value = [0, 3]
class = salmon
hour <= 8.322
entropy = 0.28
samples = 103
value = [98, 5]
class = tuna
entropy = 0.0
samples = 1
value = [0, 1]
class = salmon
entropy = 0.0
samples = 1
value = [1, 0]
class = tuna
hour <= 2.807
entropy = 0.503
samples = 27
value = [3, 24]
class = salmon
entropy = 0.0
samples = 3
value = [3, 0]
class = tuna
entropy = 0.0
samples = 1
value = [0, 1]
class = salmon
entropy = 0.0
samples = 1
value = [1, 0]
class = tuna
entropy = 0.0
samples = 4
value = [0, 4]
class = salmon
entropy = 0.0
samples = 5
value = [5, 0]
class = tuna
length <= 39.246
entropy = 0.977
samples = 34
value = [20, 14]
class = tuna
length <= 42.778
entropy = 0.161
samples = 85
value = [83, 2]
class = tuna
hour <= 8.359
entropy = 0.65
samples = 18
value = [15, 3]
class = tuna
hour <= 2.614
entropy = 0.918
samples = 9
value = [3, 6]
class = salmon
entropy = 0.0
samples = 18
value = [0, 18]
class = salmon
hour <= 4.317
entropy = 0.977
samples = 17
value = [7, 10]
class = salmon
weight <= 3.285
entropy = 0.787
samples = 17
value = [13, 4]
class = tuna
length <= 42.738
entropy = 0.65
samples = 12
value = [10, 2]
class = tuna
entropy = 0.0
samples = 73
value = [73, 0]
class = tuna
entropy = 0.0
samples = 1
value = [0, 1]
class = salmon
weight <= 3.051
entropy = 0.523
samples = 17
value = [15, 2]
class = tuna
length <= 36.673
entropy = 0.811
samples = 8
value = [2, 6]
class = salmon
entropy = 0.0
samples = 1
value = [1, 0]
class = tuna
entropy = 0.0
samples = 5
value = [0, 5]
class = salmon
length <= 38.688
entropy = 0.98
samples = 12
value = [7, 5]
class = tuna
hour <= 3.137
entropy = 1.0
samples = 6
value = [3, 3]
class = tuna
hour <= 4.519
entropy = 0.439
samples = 11
value = [10, 1]
class = tuna
length <= 42.3
entropy = 0.439
samples = 11
value = [10, 1]
class = tuna
entropy = 0.0
samples = 1
value = [0, 1]
class = salmon
weight <= 2.483
entropy = 0.918
samples = 6
value = [4, 2]
class = tuna
entropy = 0.0
samples = 11
value = [11, 0]
class = tuna
weight <= 3.543
entropy = 1.0
samples = 4
value = [2, 2]
class = tuna
entropy = 0.0
samples = 4
value = [0, 4]
class = salmon
hour <= 6.826
entropy = 0.811
samples = 8
value = [6, 2]
class = tuna
hour <= 4.911
entropy = 0.811
samples = 4
value = [1, 3]
class = salmon
entropy = 0.0
samples = 2
value = [0, 2]
class = salmon
weight <= 2.253
entropy = 0.811
samples = 4
value = [3, 1]
class = tuna
entropy = 0.0
samples = 8
value = [8, 0]
class = tuna
hour <= 5.567
entropy = 0.918
samples = 3
value = [2, 1]
class = tuna
entropy = 0.0
samples = 6
value = [6, 0]
class = tuna
length <= 42.445
entropy = 0.722
samples = 5
value = [4, 1]
class = tuna
entropy = 0.0
samples = 4
value = [4, 0]
class = tuna
entropy = 0.0
samples = 2
value = [0, 2]
class = salmon
entropy = 0.0
samples = 2
value = [2, 0]
class = tuna
entropy = 0.0
samples = 4
value = [4, 0]
class = tuna
length <= 37.87
entropy = 1.0
samples = 4
value = [2, 2]
class = tuna
entropy = 0.0
samples = 1
value = [1, 0]
class = tuna
entropy = 0.0
samples = 3
value = [0, 3]
class = salmon
entropy = 0.0
samples = 1
value = [1, 0]
class = salmon
entropy = 0.0
samples = 3
value = [3, 0]
class = tuna
entropy = 0.0
samples = 1
value = [0, 1]
class = salmon
entropy = 0.0
samples = 2
value = [2, 0]
class = tuna
entropy = 0.0
samples = 1
value = [0, 1]
class = salmon
weight <= 3.016
entropy = 0.918
samples = 3
value = [2, 1]
class = tuna
entropy = 0.0
samples = 1
value = [0, 1]
class = salmon
entropy = 0.0
samples = 2
value = [2, 0]
class = tuna


length <= 37.533
entropy = 0.999
samples = 600
value = [311, 289]
class = tuna
True
False
weight <= 5.948
entropy = 0.401
samples = 289
value = [23, 266]
class = salmon
weight <= 5.8
entropy = 0.381
samples = 311
value = [288, 23]
class = tuna
length <= 35.172
entropy = 0.21
samples = 271
value = [9, 262]
class = salmon
length <= 34.306
entropy = 0.754
samples = 18
value = [14, 4]
class = tuna
length <= 41.499
entropy = 0.628
samples = 146
value = [123, 23]
class = tuna
entropy = 0.0
samples = 165
value = [165, 0]
class = tuna
weight <= 1.648
entropy = 0.07
samples = 239
value = [2, 237]
class = salmon
weight <= 5.107
entropy = 0.758
samples = 32
value = [7, 25]
class = salmon
weight <= 7.781
entropy = 1.0
samples = 8
value = [4, 4]
class = tuna
entropy = 0.0
samples = 10
value = [10, 0]
class = tuna
length <= 41.08
entropy = 0.974
samples = 42
value = [25, 17]
class = tuna
weight <= 5.734
entropy = 0.318
samples = 104
value = [98, 6]
class = tuna
entropy = 0.0
samples = 2
value = [2, 0]
class = tuna
entropy = 0.0
samples = 237
value = [0, 237]
class = salmon
weight <= 1.055
entropy = 0.592
samples = 28
value = [4, 24]
class = salmon
weight <= 5.772
entropy = 0.811
samples = 4
value = [3, 1]
class = tuna
weight <= 6.016
entropy = 0.722
samples = 5
value = [1, 4]
class = salmon
entropy = 0.0
samples = 3
value = [3, 0]
class = tuna
weight <= 2.162
entropy = 0.942
samples = 39
value = [25, 14]
class = tuna
entropy = 0.0
samples = 3
value = [0, 3]
class = salmon
hour <= 8.322
entropy = 0.28
samples = 103
value = [98, 5]
class = tuna
entropy = 0.0
samples = 1
value = [0, 1]
class = salmon
entropy = 0.0
samples = 1
value = [1, 0]
class = tuna
hour <= 2.807
entropy = 0.503
samples = 27
value = [3, 24]
class = salmon
entropy = 0.0
samples = 3
value = [3, 0]
class = tuna
entropy = 0.0
samples = 1
value = [0, 1]
class = salmon
entropy = 0.0
samples = 1
value = [1, 0]
class = tuna
entropy = 0.0
samples = 4
value = [0, 4]
class = salmon
entropy = 0.0
samples = 5
value = [5, 0]
class = tuna
length <= 39.246
entropy = 0.977
samples = 34
value = [20, 14]
class = tuna
length <= 42.778
entropy = 0.161
samples = 85
value = [83, 2]
class = tuna
hour <= 8.359
entropy = 0.65
samples = 18
value = [15, 3]
class = tuna
hour <= 2.614
entropy = 0.918
samples = 9
value = [3, 6]
class = salmon
entropy = 0.0
samples = 18
value = [0, 18]
class = salmon
hour <= 4.317
entropy = 0.977
samples = 17
value = [7, 10]
class = salmon
weight <= 3.285
entropy = 0.787
samples = 17
value = [13, 4]
class = tuna
length <= 42.738
entropy = 0.65
samples = 12
value = [10, 2]
class = tuna
entropy = 0.0
samples = 73
value = [73, 0]
class = tuna
entropy = 0.0
samples = 1
value = [0, 1]
class = salmon
weight <= 3.051
entropy = 0.523
samples = 17
value = [15, 2]
class = tuna
hour <= 1.818
entropy = 0.811
samples = 8
value = [2, 6]
class = salmon
entropy = 0.0
samples = 1
value = [1, 0]
class = tuna
entropy = 0.0
samples = 5
value = [0, 5]
class = salmon
length <= 38.688
entropy = 0.98
samples = 12
value = [7, 5]
class = tuna
hour <= 3.137
entropy = 1.0
samples = 6
value = [3, 3]
class = tuna
hour <= 4.519
entropy = 0.439
samples = 11
value = [10, 1]
class = tuna
hour <= 4.447
entropy = 0.439
samples = 11
value = [10, 1]
class = tuna
entropy = 0.0
samples = 1
value = [0, 1]
class = salmon
weight <= 2.483
entropy = 0.918
samples = 6
value = [4, 2]
class = tuna
entropy = 0.0
samples = 11
value = [11, 0]
class = tuna
hour <= 1.195
entropy = 1.0
samples = 4
value = [2, 2]
class = tuna
entropy = 0.0
samples = 4
value = [0, 4]
class = salmon
hour <= 6.826
entropy = 0.811
samples = 8
value = [6, 2]
class = tuna
hour <= 4.911
entropy = 0.811
samples = 4
value = [1, 3]
class = salmon
entropy = 0.0
samples = 2
value = [0, 2]
class = salmon
weight <= 2.253
entropy = 0.811
samples = 4
value = [3, 1]
class = tuna
entropy = 0.0
samples = 8
value = [8, 0]
class = tuna
weight <= 4.215
entropy = 0.918
samples = 3
value = [2, 1]
class = tuna
hour <= 3.303
entropy = 0.722
samples = 5
value = [4, 1]
class = tuna
entropy = 0.0
samples = 6
value = [6, 0]
class = tuna
entropy = 0.0
samples = 4
value = [4, 0]
class = tuna
entropy = 0.0
samples = 2
value = [0, 2]
class = salmon
entropy = 0.0
samples = 2
value = [2, 0]
class = salmon
entropy = 0.0
samples = 4
value = [4, 0]
class = tuna
hour <= 7.296
entropy = 1.0
samples = 4
value = [2, 2]
class = tuna
entropy = 0.0
samples = 1
value = [1, 0]
class = tuna
entropy = 0.0
samples = 3
value = [0, 3]
class = salmon
entropy = 0.0
samples = 1
value = [1, 0]
class = tuna
entropy = 0.0
samples = 3
value = [3, 0]
class = salmon
entropy = 0.0
samples = 2
value = [2, 0]
class = tuna
entropy = 0.0
samples = 1
value = [0, 1]
class = tuna
entropy = 0.0
samples = 1
value = [0, 1]
class = salmon
weight <= 3.016
entropy = 0.918
samples = 3
value = [2, 1]
class = tuna
entropy = 0.0
samples = 1
value = [0, 1]
class = salmon
entropy = 0.0
samples = 2
value = [2, 0]
class = tuna

Overfitting : Iris dataset, max depth=34



Avoiding overfitting : Iris dataset

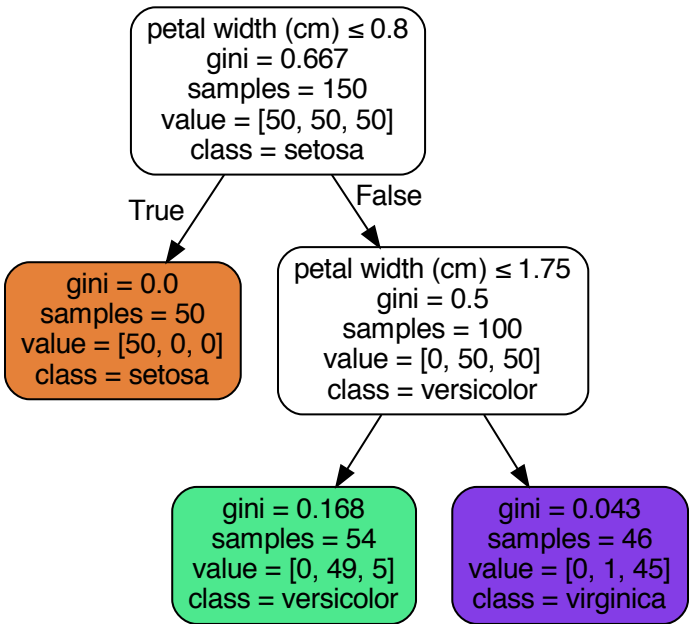


Figure – Avoid overfitting with pre-pruning. In this case, impose a minimum impurity decrease of 0.1

Post-pruning

It is also possible to prune after the construction of a large tree.

Remarks on decision trees I

- Decision trees do not require hypotheses on the distribution of de features.
- Feature selection is integrated in the algorithm, so they might be relevant in a situation with many features (variables explicatives)
- A segmentation is invariant to a monotonic change in a feature. It is thus robust to outliers or to very asymetrical distributions.
- Decision trees allow a straightforward **interpretation** of the model.

Remarks on decision trees II

- Decision trees are greedy : they might find a local minimum
- they are hierarchical : a bad choice in a segmentation at the top of the tree propagates in the whole tree
- Hence their instability and sensibility to the choice of the training set.

Ensemble learning

- *Bagging* and *boosting* are methods that combine estimators (in parallel or sequentially)
- they are often applied to decision trees
- they reach state-of-the-art performance in several supervised learning tasks [Fernández-Delgado et al., 2014]
- they are an active area of research (both theoretically and for practical applications).

`https://scikit-learn.org/stable/modules/ensemble.html`

Aggregating to reduce the variance

- usual setup : input space $\mathcal{X}$, output space $\mathcal{Y}$.
- we note z_b a dataset sampled from the unknown distribution ρ. If we sample b different datasets, $b \in [1, B]$,

$$z_b = \{(x_{1b}, y_{1b}), \dots (x_{nb}, y_{nb})\} \tag{25}$$

- we note $\hat{f_{z_b}}$ the estimator obtained after learning with z_b.

Aggregate to reduce the variance

- we note z_b a dataset sampled from the unknown distribution ρ. If we sample b different datasets, $b \in [1, B]$,

$$z_b = \{(x_{1b}, y_{1b}), \dots (x_{nb}, y_{nb})\} \tag{26}$$

- we note $\hat{f}_{z_b}$ the estimator obtained after learning with z_b.

Aggregating consists in using as an estimator :

- for regression

$$\hat{f}_B = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_{z_b} \tag{27}$$

- for classification

$$\hat{f}_B(x) = \arg\max_j |\{b, \hat{f}_{z_b}(x) = j\}| \tag{28}$$

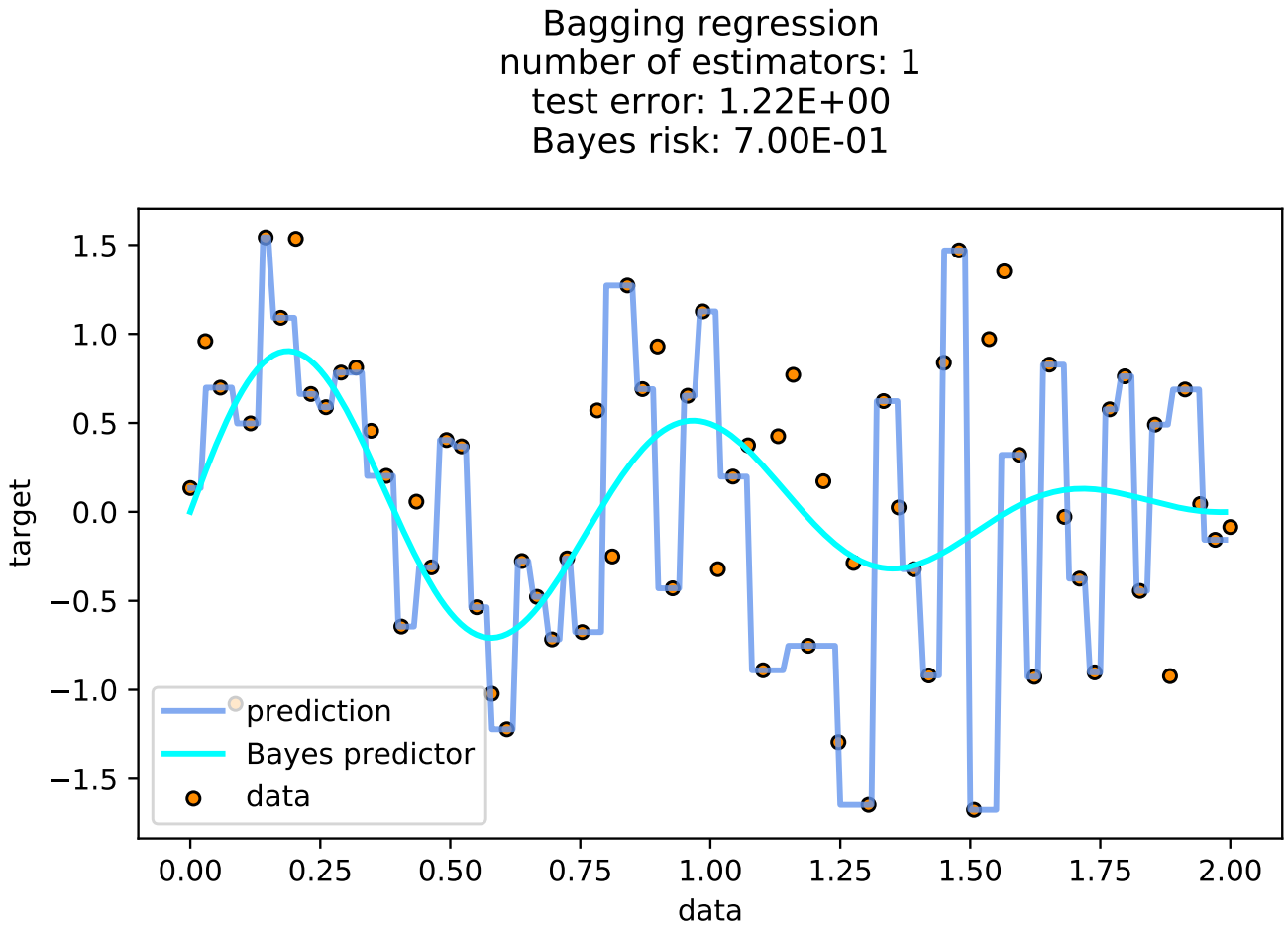
Bootstrapping and bagging

If B is large, it is not possible to sample B independent datasets with n samples, from a finite dataset.
Bootstrapping consists in sampling B times a sample dataset with n elements **with replacement**.
Bagging is the combination of bootstrapping and aggregating.

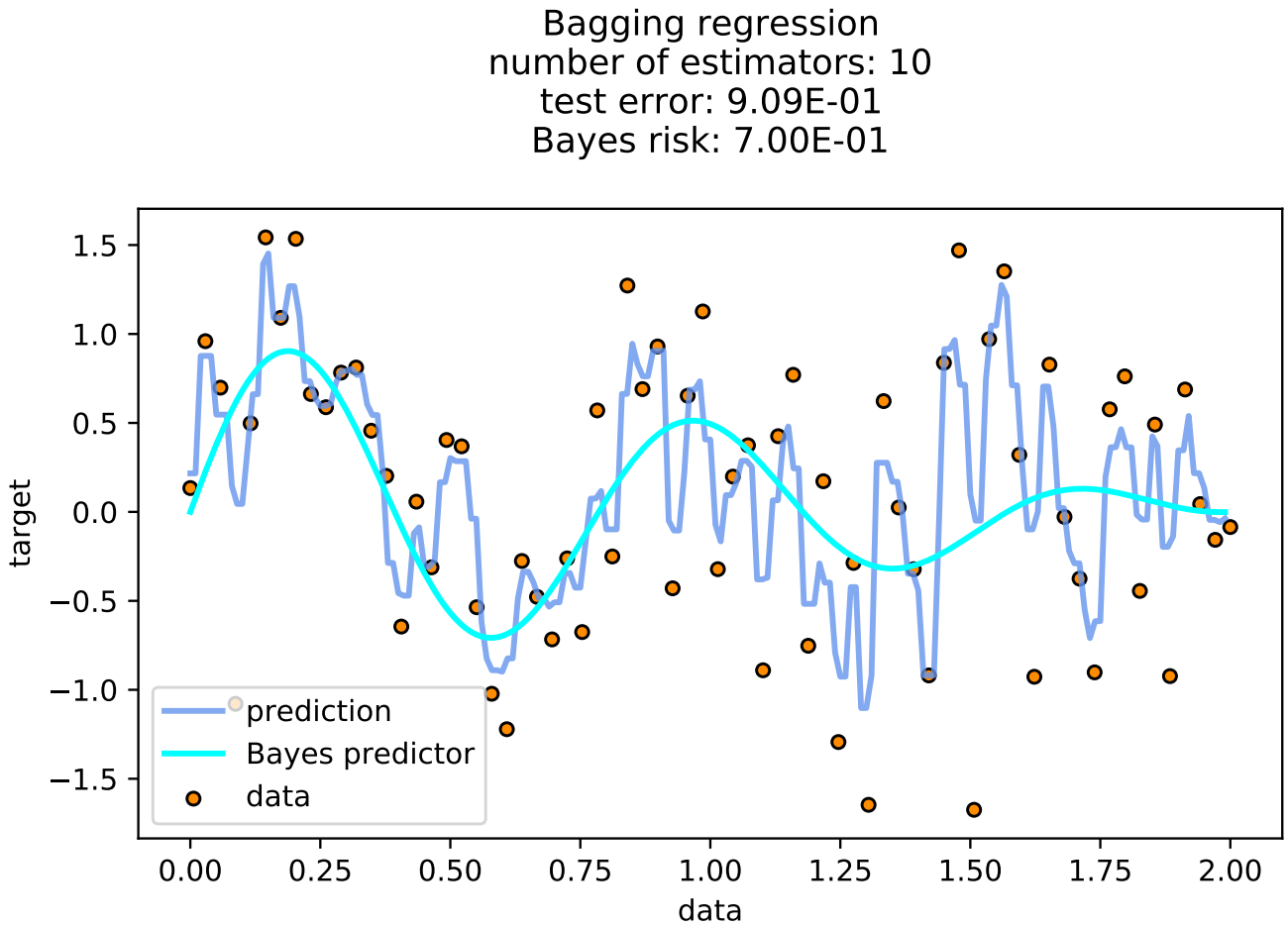
Out-of-bag error

Vocabulary : for an observation (x_i, y_i), the **out-of-bag error** is the mean error on this observation among the estimators that were trained on a bootstrap dataset **not** containing it.

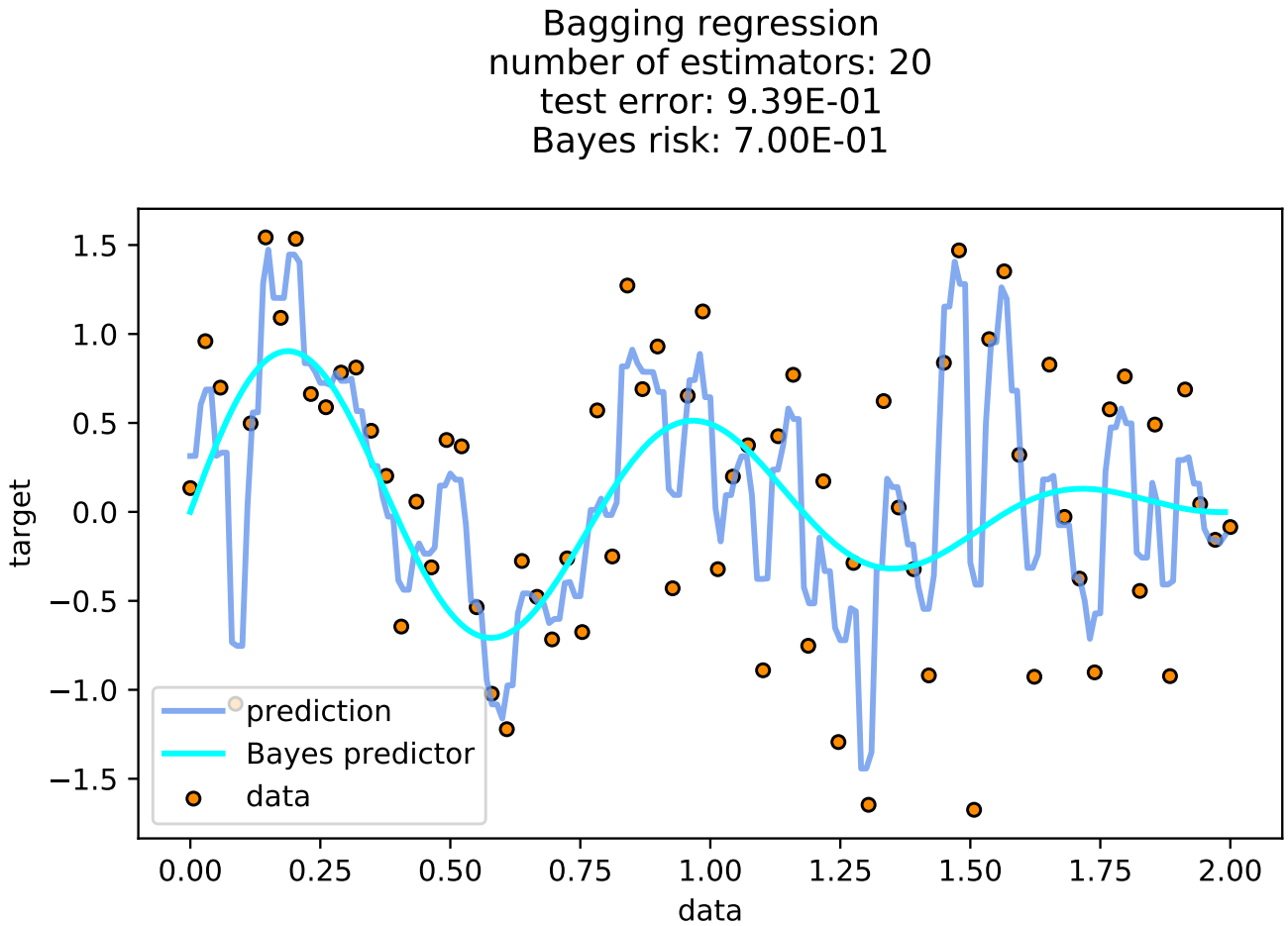
Simulation


Bagging regression
number of estimators: 1
test error: 1.22E+00
Bayes risk: 7.00E-01
target
data
prediction
Bayes predictor
data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

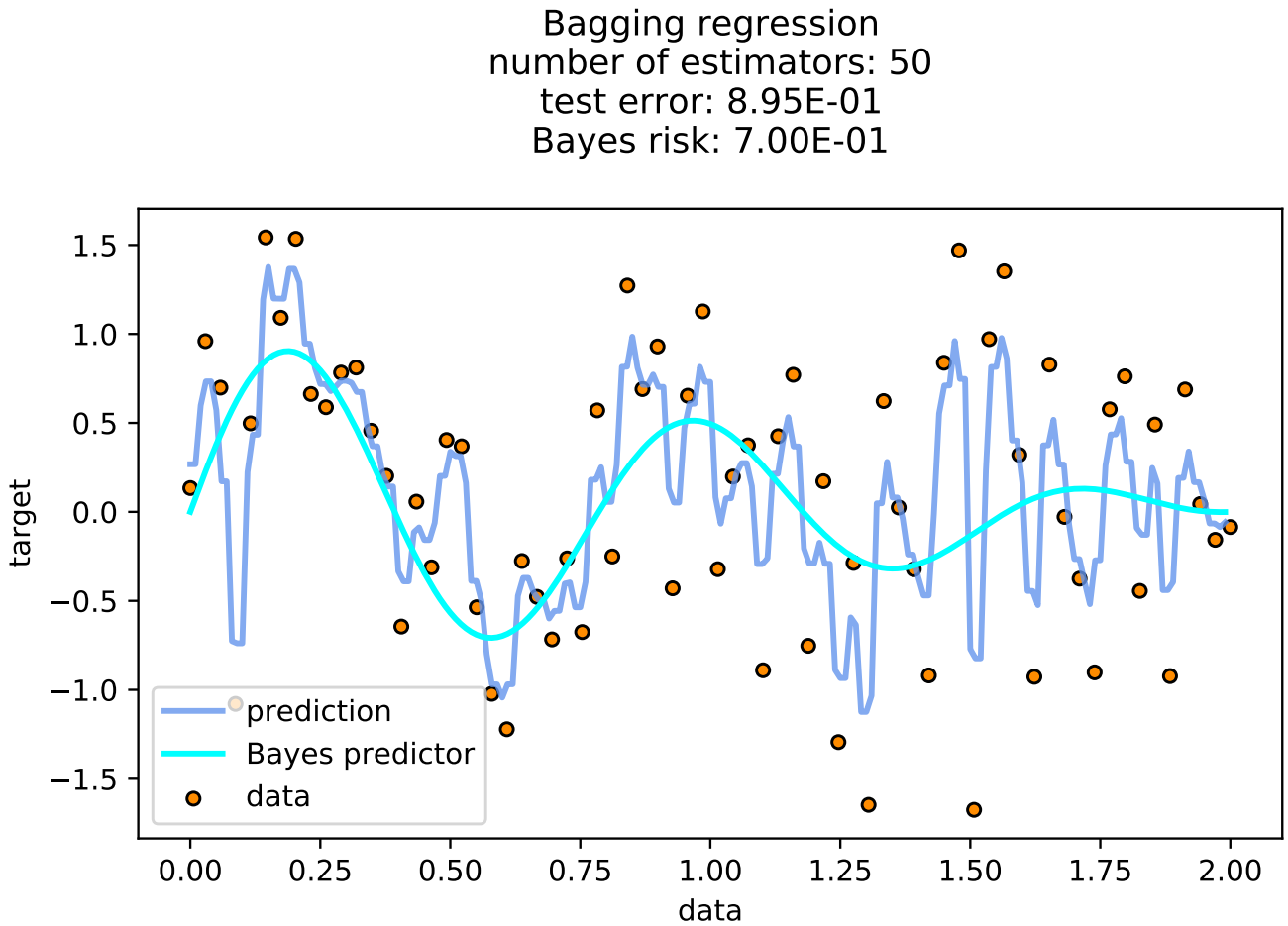
Simulation


Bagging regression
number of estimators: 10
test error: 9.09E-01
Bayes risk: 7.00E-01
target
data
prediction
Bayes predictor
data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

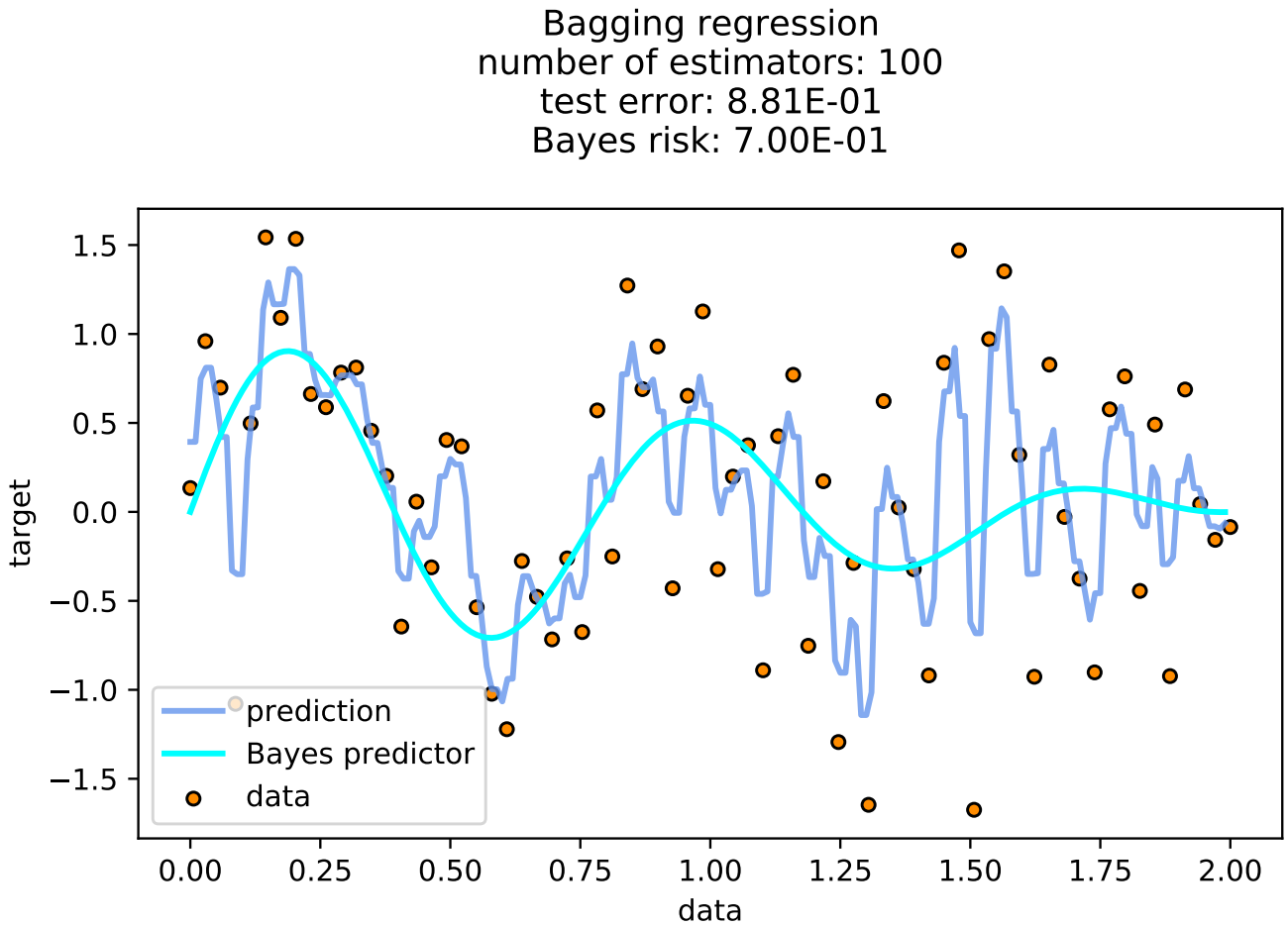
Simulation


Bagging regression
number of estimators: 20
test error: 9.39E-01
Bayes risk: 7.00E-01
target
data
prediction
Bayes predictor
data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Simulation


Bagging regression
number of estimators: 50
test error: 8.95E-01
Bayes risk: 7.00E-01
target
data
prediction
Bayes predictor
data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Simulation


Bagging regression
number of estimators: 100
test error: 8.81E-01
Bayes risk: 7.00E-01
target
data
prediction
Bayes predictor
data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Individual estimators

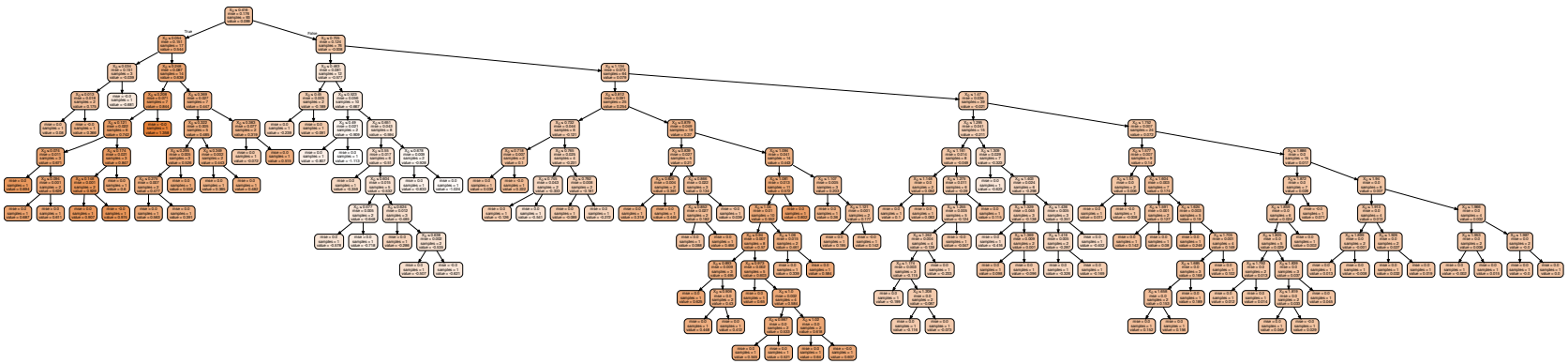


Figure – Estimator used in the averaging

Individual estimators

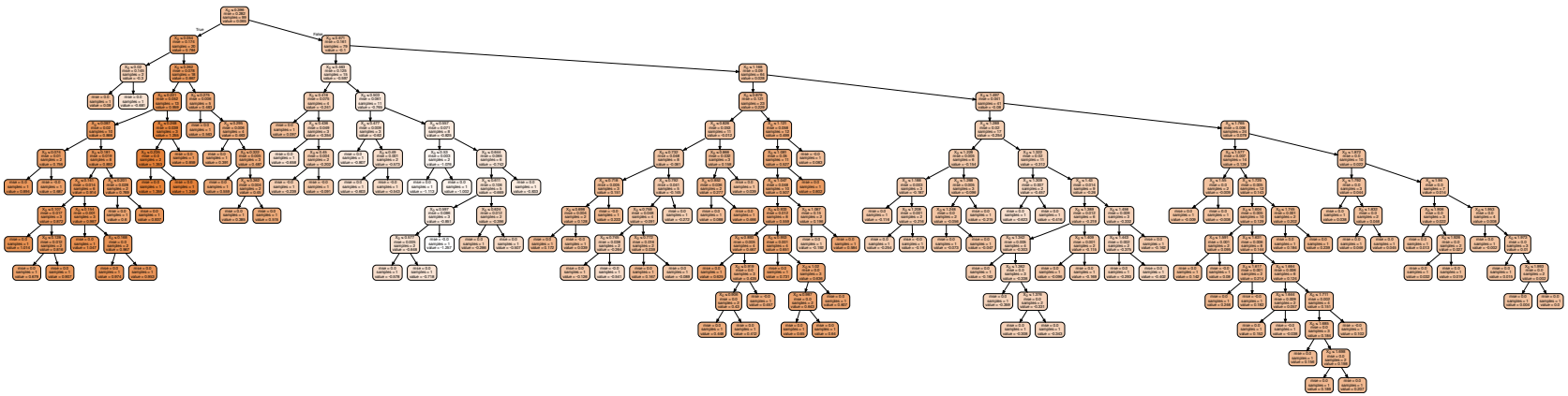


Figure – Other estimator used in the averaging

Possible issues

- number of trees to compute (B) in order to have a stable out-of-bag error.
- need for storing all the base estimators
- the final estimator is a *black-box* model.

Random forest

- Random forest [Breiman, 2001] is a bagging method with binary CART estimators, with additional randomness added to the choice of segmentation variables.
- the goal is to increase the **independence** between the base trees.
- although the theoretical properties of random forest are not yet fully understood, it is an important benchmark in supervised learning (but for instance not when the problem can be solved in a linear way).

Variance of an average : independent variables

Recall that if $(X_k)_{k\in\mathbb{N}}$ is a sequence of i.i.d. real variables that have a moment of order 2, and hence a variance σ^2, and an expected value of m. Then if $S_B = \frac{1}{B}\sum_{i=1}^{B} X_i$

$$\begin{aligned} Var(S_B) &= \sum_{i=1}^{B} Var(\frac{1}{B}X_i) \\ &= \frac{1}{B^2}\sum_{i=1}^{B} Var(X_i) \\ &= \frac{\sigma^2}{B} \end{aligned} \tag{29}$$

Variance of the average : correlated variables

However, if the X_i are identically distributed but correlated with correlation c, then we admit that

$$Vac(S_B) = c\sigma^2 + \frac{1-c}{B}\sigma^2 \tag{30}$$

Random forest diminishes c by adding some randomness in the choice of the segmentation variables.

Random forest

Let d be the number of features. Let $m \leq d$ be an integer.

Result: Random forest estimator

for $b \in [1, B]$ **do**

  Sample a bootstrap dataset z_b;

  Estimate $\hat{f}_{z_b}$ with **randomization** of the variables. The search of the optimal segmentation is done on m randomly sampled variables.;

end

$$\hat{f}_B = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_{z_b} \tag{31}$$

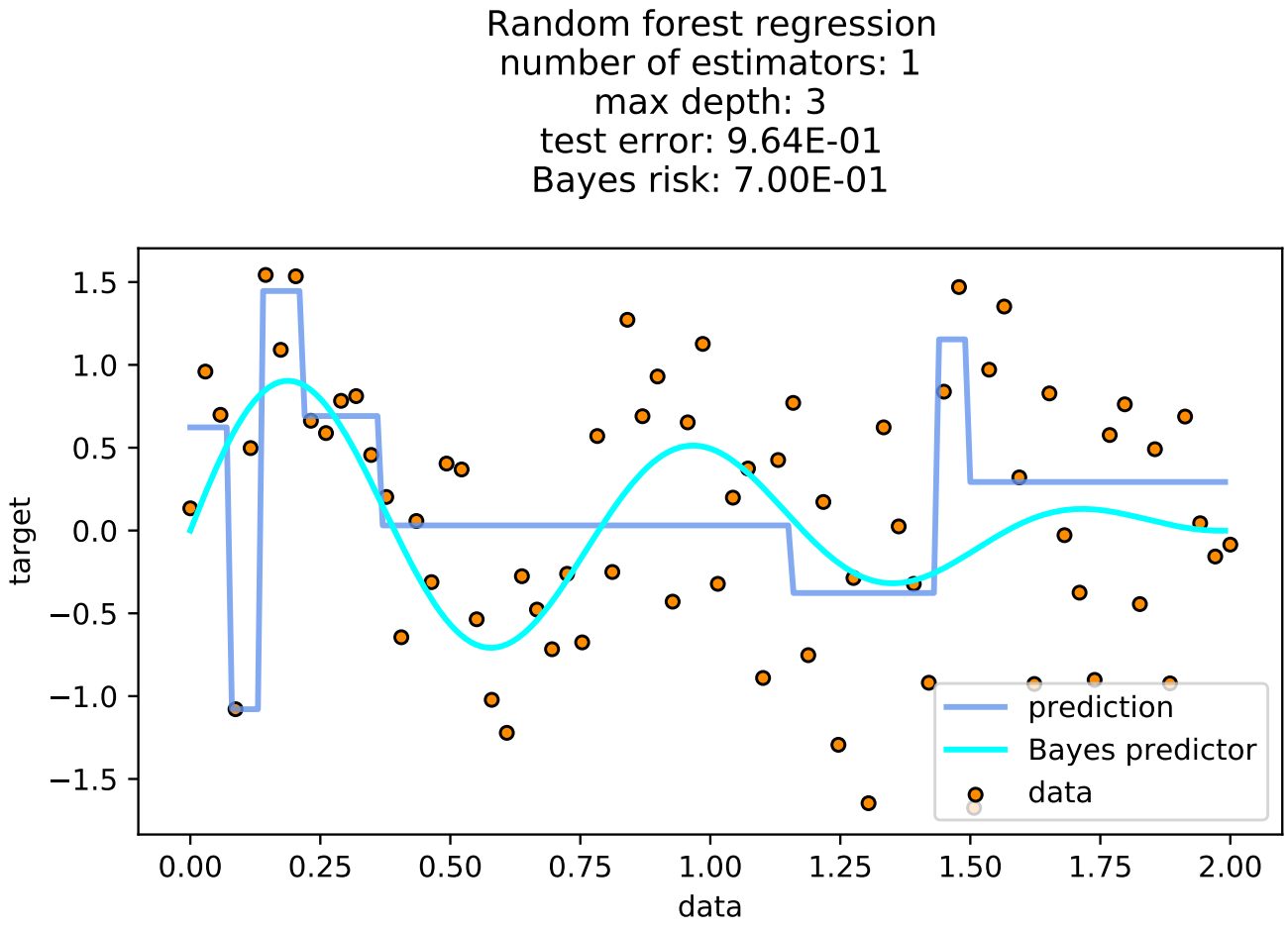
return $\hat{f}_B$

Algorithm 1: Random forest

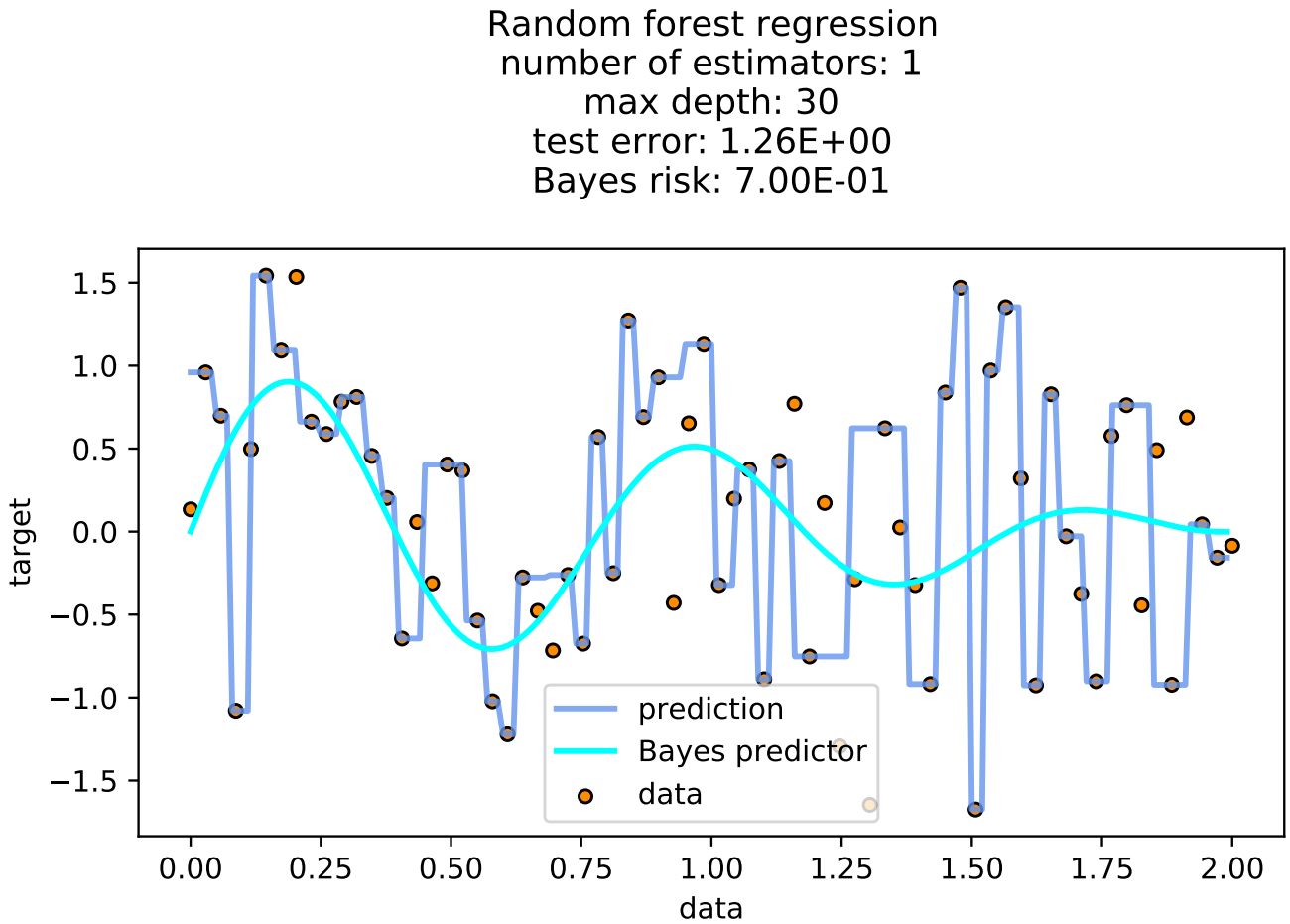
Remarks on random forest

- With random forest, it is possible to use a more brutal pruning (for instance using only trees of depth 2)
- To tune m, heuristics are used. Common ones include :
  - $m = \sqrt{d}$ for classification (d is the number of features)
  - $m = d/3$ for regression
  - cross validation.

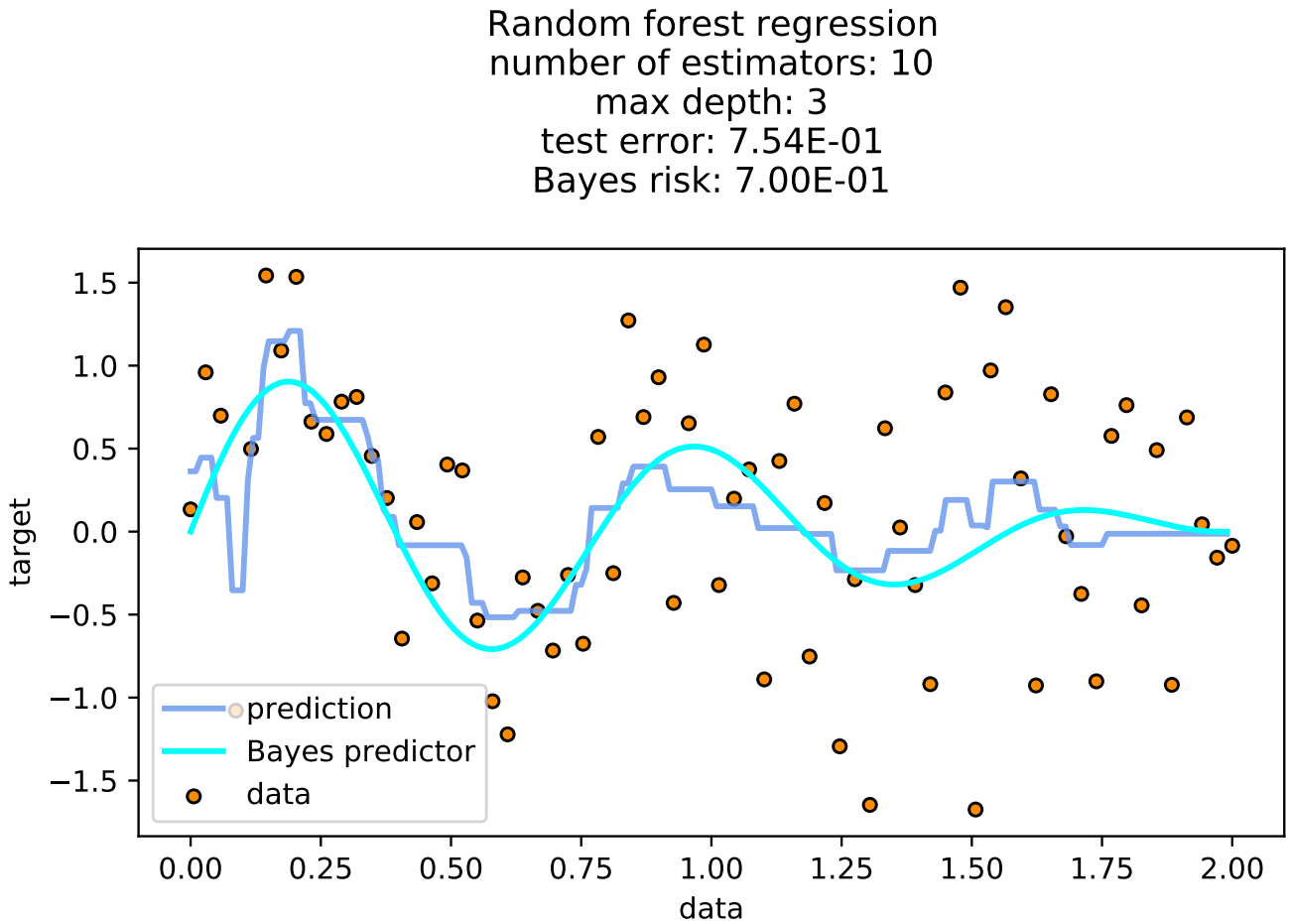
Example in 1D



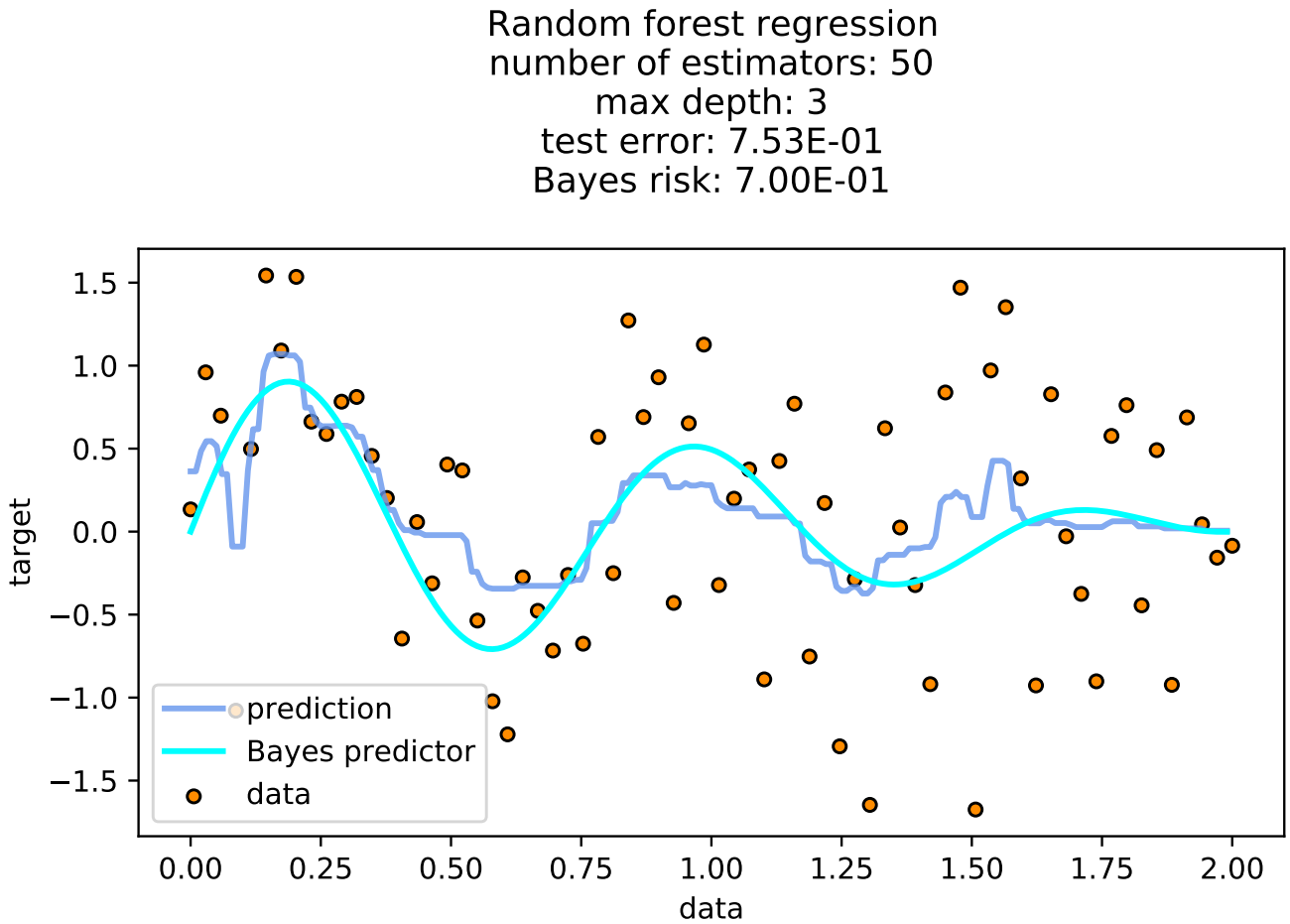
Example in 1D



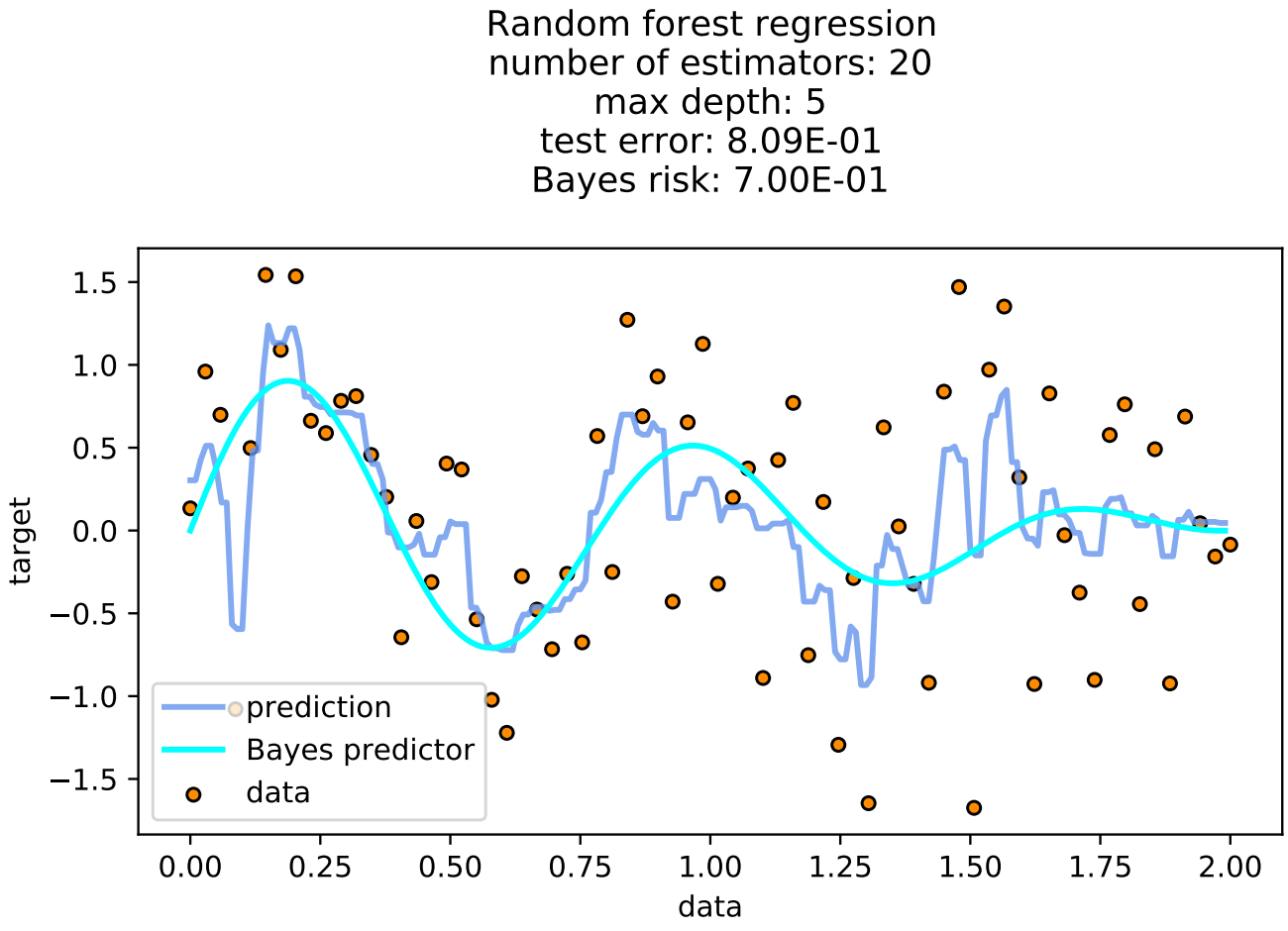
Example in 1D



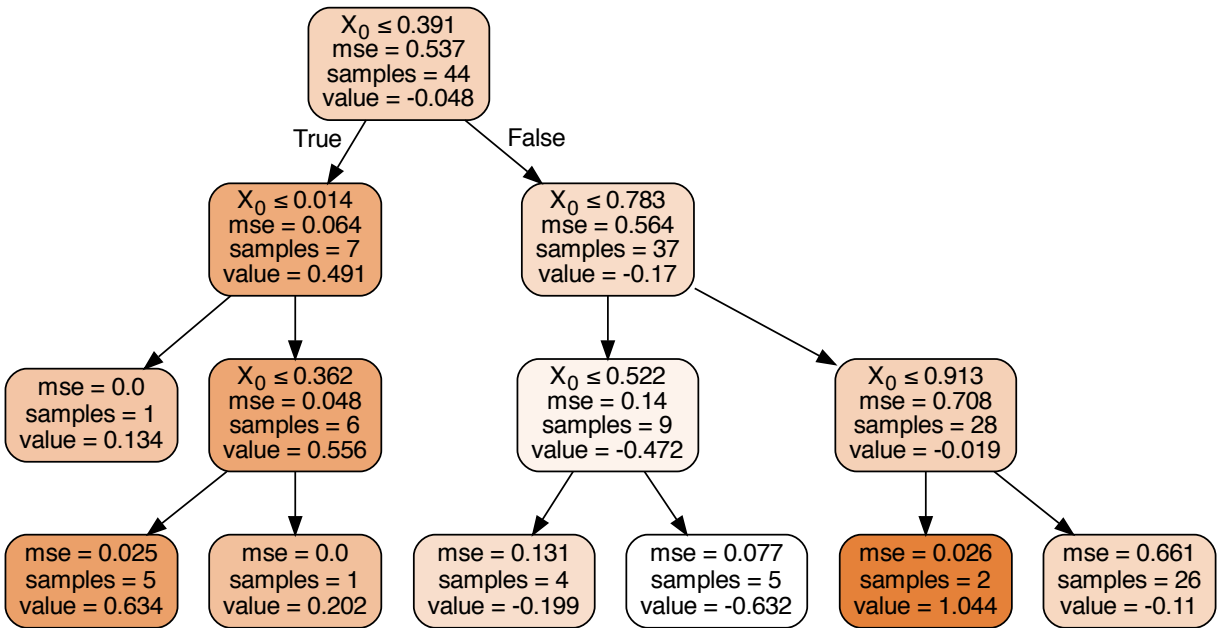
Example in 1D



Example in 1D



Estimators


$X_0 \leq 0.391$
mse = 0.537
samples = 44
value = -0.048
True
False
$X_0 \leq 0.014$
mse = 0.064
samples = 7
value = 0.491
$X_0 \leq 0.783$
mse = 0.564
samples = 37
value = -0.17
mse = 0.0
samples = 1
value = 0.134
$X_0 \leq 0.362$
mse = 0.048
samples = 6
value = 0.556
$X_0 \leq 0.522$
mse = 0.14
samples = 9
value = -0.472
$X_0 \leq 0.913$
mse = 0.708
samples = 28
value = -0.019
mse = 0.025
samples = 5
value = 0.634
mse = 0.0
samples = 1
value = 0.202
mse = 0.131
samples = 4
value = -0.199
mse = 0.077
samples = 5
value = -0.632
mse = 0.026
samples = 2
value = 1.044
mse = 0.661
samples = 26
value = -0.11

Importance of the variables

Although the obtained estimator is hard to interpret, we can still study the statistical importance of variables for the prediction.

Mean decrease accuracy

Consider a variable j. For a given tree b

- compute the out-of-bag error.
- shuffle the values of j in the out-of-bag sample.
- compute the new out-of-bag error
- store the decrease in prediction quality, D_j^b.

Average the D_j^b on b in order to measure the importance of the variable j.

Boosting

- While bagging is a random method, **boosting** is an adaptive method.
- Boosting builds on *weak classifiers*, and like bagging, aggregates them, for instance with a **weighted sum**.
- However, the weak classifiers are built **sequentially**. The classifier $b+1$ adapts the classifier b by focusing on improving the prediction of incorrectly classified training samples.
- Several variants exists (in the weighting, loss function, aggregating method, etc).

Adaboost

- Original boosting algorithm [Freund and Schapire, 1996]
- $\mathcal{Y} = \{-1, 1\}$ (but can also be adapted to a regression problem)
- Given the sample dataset $z = \{(x_1, y_1), \dots (x_n, y_n)\}$. We learn a **sequence of estimators** $(\delta_m)_{m \in [1..B]}$ (often CART).

Initialize the weights $w = \{w_i = \frac{1}{n}, i = 1..n\}$;
for $m = 1..B$ **do**

Estimate δ_m on the weighted dataset. Compute the error rate

$$\hat{\epsilon_m} = \frac{\sum_{i=1}^{n} w_i 1(\delta_m(x_i) \neq y_i)}{\sum_{i=1}^{n} w_i} \tag{32}$$

Compute the logit of $\hat{\epsilon_m}$

$$c_m = \log\left(\frac{1 - \hat{\epsilon_m}}{\hat{\epsilon_m}}\right) \tag{33}$$

Update the weights

$$w_i \leftarrow w_i \exp\Big(c_m 1(\delta_m(x_i) \neq y_i)\Big) \tag{34}$$

end

$$\hat{\delta_B} = \text{sign}\Big(\sum_{b=1}^{B} \delta_{z_b}\Big) \tag{35}$$

return $\hat{f_B}$

Algorithm 2: Adaboost for binary classification (adaptive boos-

Remarks

We need to enforce that $c_m \geq 0$. It is verified if $\hat{\epsilon_m} \geq \frac{1}{2}$.

$$c_m = \log\left(\frac{1 - \hat{\epsilon_m}}{\hat{\epsilon_m}}\right) \tag{36}$$

- It is experimentally shown that if the weak classifiers are *stump trees* (2 leaves), then adaBoost performs better than one given tree with a number of leaves equal to the number of iterations of adaBoost (hence a comparable computation time).
- A number of leaves q between 4 and 8 for the weak classifiers is often recommended.
- adaBoost can be adapted to multiclass classification and regression [Schapire, 2003].

Gradient tree boosting

- differentiate with respect to the predictions of the tree (compute a gradient)
- approximate the gradient by a regression tree
- update the tree following this gradient.

Gradient tree boosting

Initialize $f_0 = \arg\min_z \sum_{i=1}^{n} l(y_i, z)$;
for $m = 1..B$ **do**
  Compute $r_{m_i} = -\frac{\partial l(y_i, f(x_i))}{\partial f(x_i)}(f = f_{m-1}), i \in [1, n]$
  Train a decision tree δ_m on $(x_i, r_{m_i})_{i \in [1,n]}$
  Compute γ_m minimizing

$$\sum_{i=1}^{n} l(y_i, f_{m-1}(x) + \gamma \delta_m(x_i)) \tag{37}$$

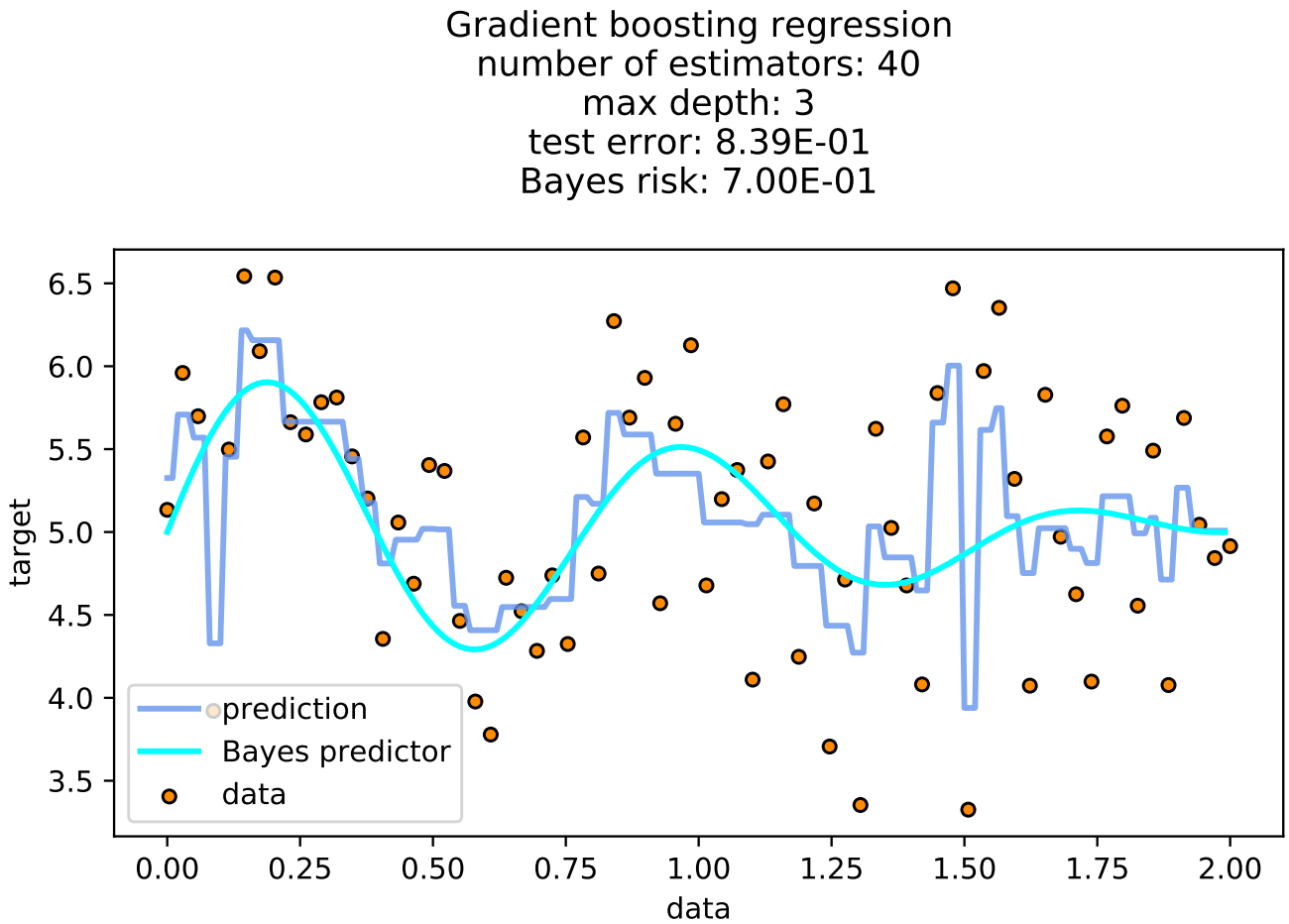
  Update the estimator

$$\hat{f_m} = \hat{f_{m-1}} + \gamma_m \delta_m \tag{38}$$

end
return $\hat{f_B}$

Algorithm 3: Gradient tree boosting

Simulation


Gradient boosting regression
number of estimators: 40
max depth: 3
test error: 8.39E-01
Bayes risk: 7.00E-01
target
data
prediction
Bayes predictor
data
6.5
6.0
5.5
5.0
4.5
4.0
3.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Gradient tree boosting

Many parameter can be tuned :

- maximum depth of the estimators
- shrinkage η ($\hat{f_m} = \hat{f_{m-1}} + \eta\gamma_m\delta_m$, with $0 \leq \eta \leq 1$). If η is low, e.g. ≤ 0.1, it can lead to a slower convergence but might prevent overfitting.
- number of trees learned B

`https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-pytho`

It is necessary to experiment, read documentations, cross validate, use heuristics, etc.

Extrem gradient boosting

- XGBoost : variant of gradient boosting, very efficient on several benchmarks [Chen and Guestrin, 2016]
- can exploit parallelization
- `https://xgboost.readthedocs.io/en/latest/parameter.html`
- `https://github.com/dmlc/xgboost`

Catboost

- See also : Catboost
  `https://catboost.ai/`

References I


Bottou, L. and Bousquet, O. (2009).
The tradeoffs of large scale learning.
Advances in Neural Information Processing Systems 20 - Proceedings of the 2007 Conference, (January 2007).


Breiman, L. (2001).
Random Forests.
Machine Learning, 45(1) :5–32.


Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (2017).
Classification And Regression Trees.
Routledge.

References II

Chen, T. and Guestrin, C. (2016).
XGBoost : A scalable tree boosting system.
Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 13-17-Augu :785–794.

Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. (2014).
Do we need hundreds of classifiers to solve real world classification problems ?
Journal of Machine Learning Research, 15 :3133–3181.

Freund, Y. and Schapire, R. E. (1996).
Experiments with a New Boosting Algorithm.
Proceedings of the 13th International Conference on Machine Learning, pages 148–156.

References III

Schapire, R. E. (2003).
The Boosting Approach to Machine Learning : An Overview
BT - Nonlinear Estimation and Classification.
Nonlinear Estimation and Classification, 171(Chapter 9) :149–171.