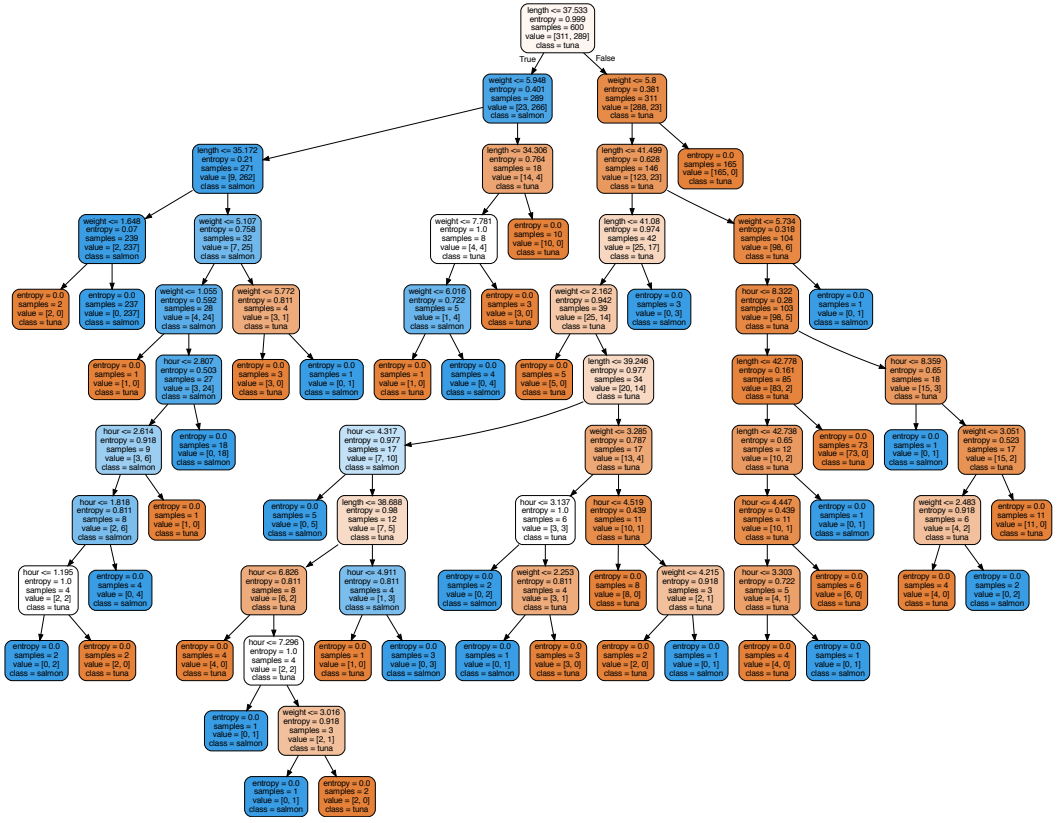
Fondamentaux théoriques du machine learning



Risks and risk decompositions

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

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

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, $\theta \in \mathbb{R}^m$.

$$\forall x, f(x) = \langle \theta, \phi(x) \rangle \tag{10}$$

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, $\theta \in \mathbb{R}^m$.

$$\forall x, f(x) = \langle \theta, \phi(x) \rangle \tag{11}$$

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

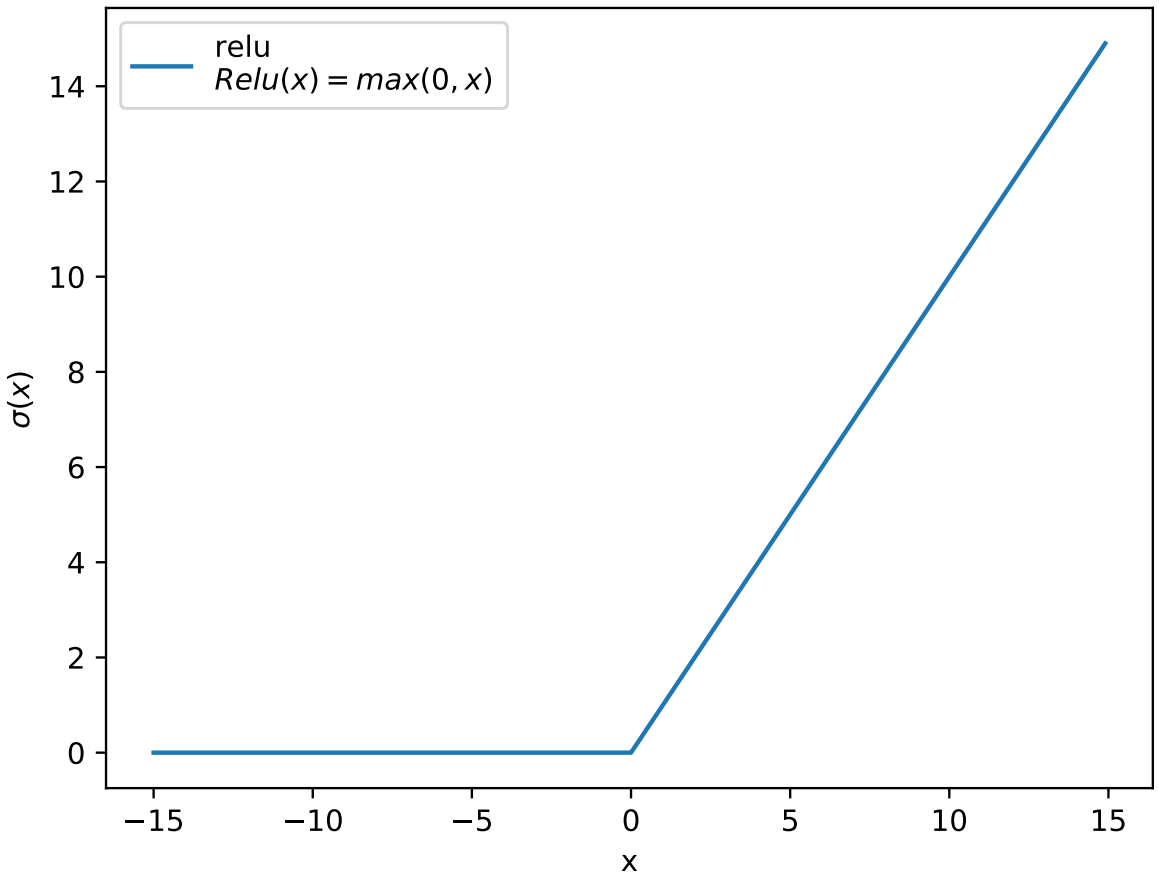
In order to learn a multidimensional output of dimension p ($\mathcal{Y} = \mathbb{R}^p$) , it is sufficient to lean a matrix $\theta \in \mathbb{R}^{m,p}$.

Single layer neural network

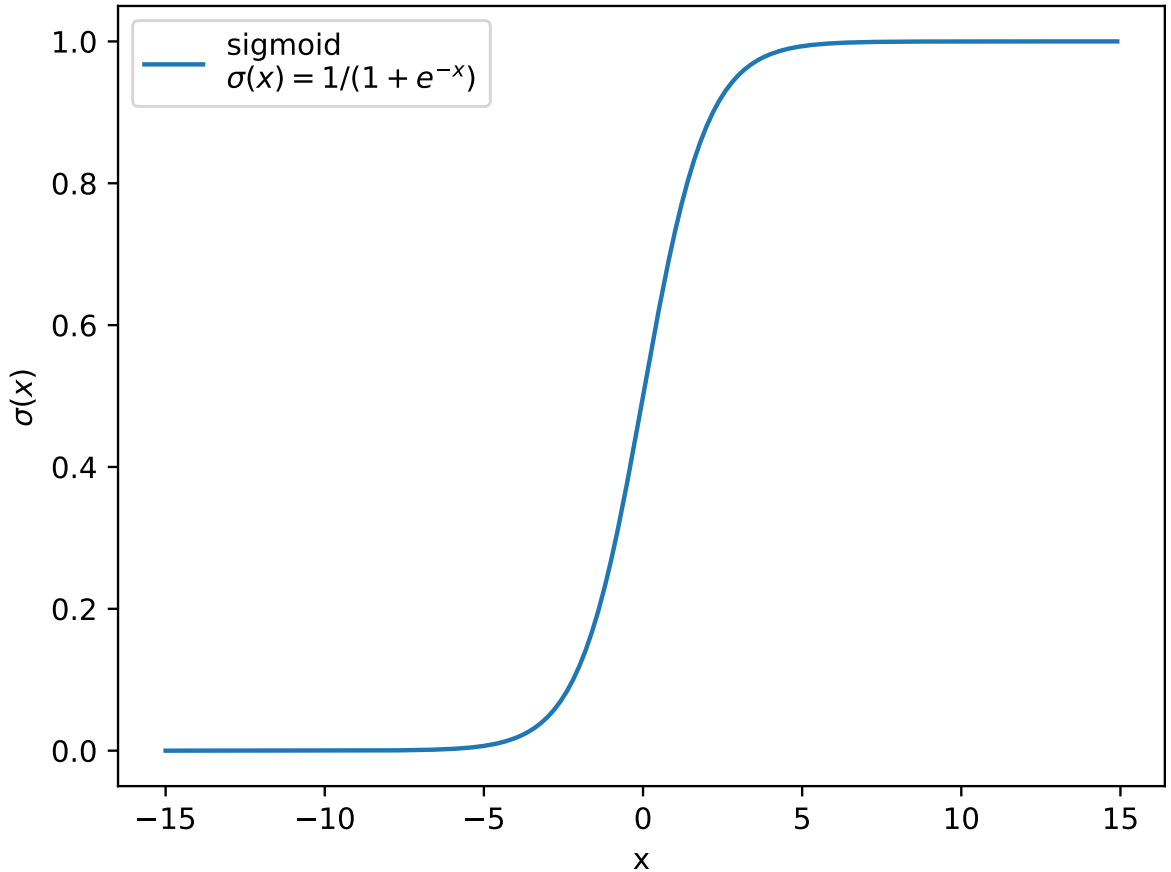
$$f(x) = \sum_{j=1}^{m} \theta_j \sigma(w_j^T x + b_j) \tag{12}$$

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

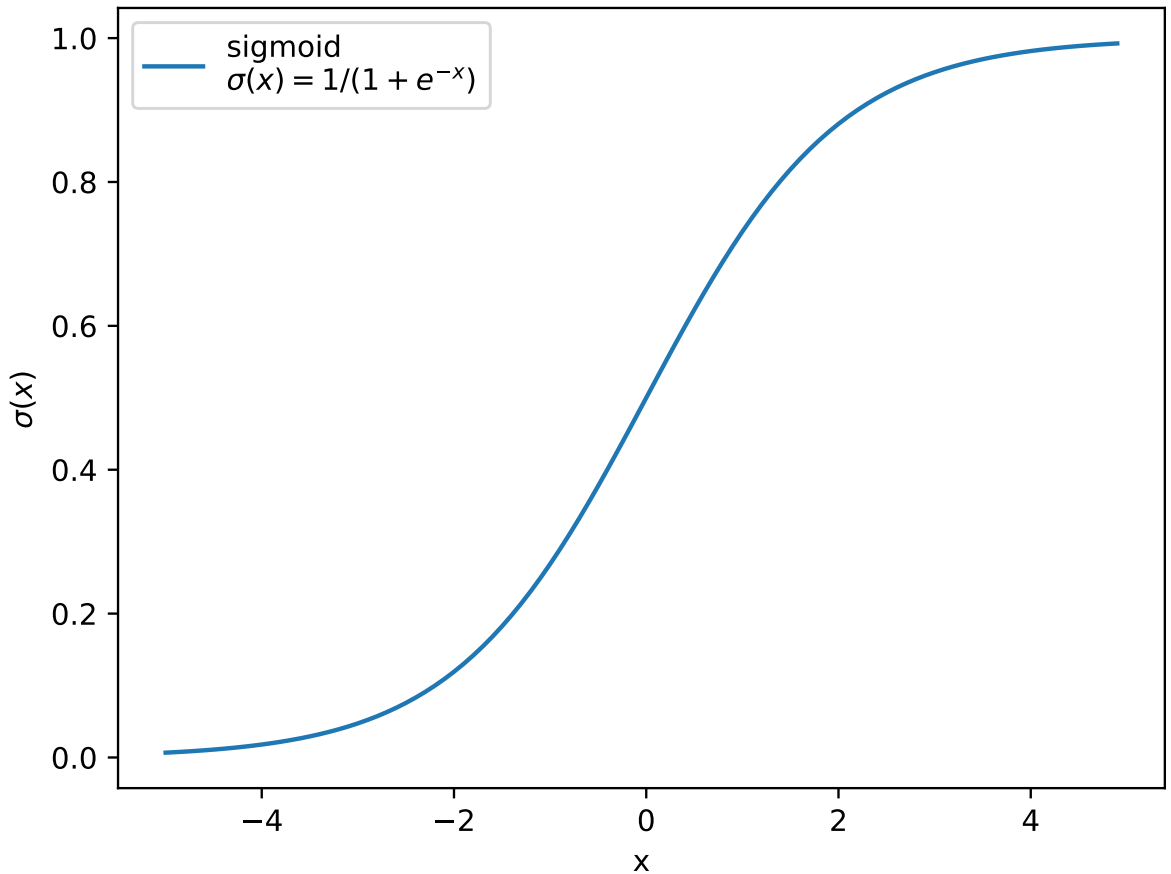
ReLU



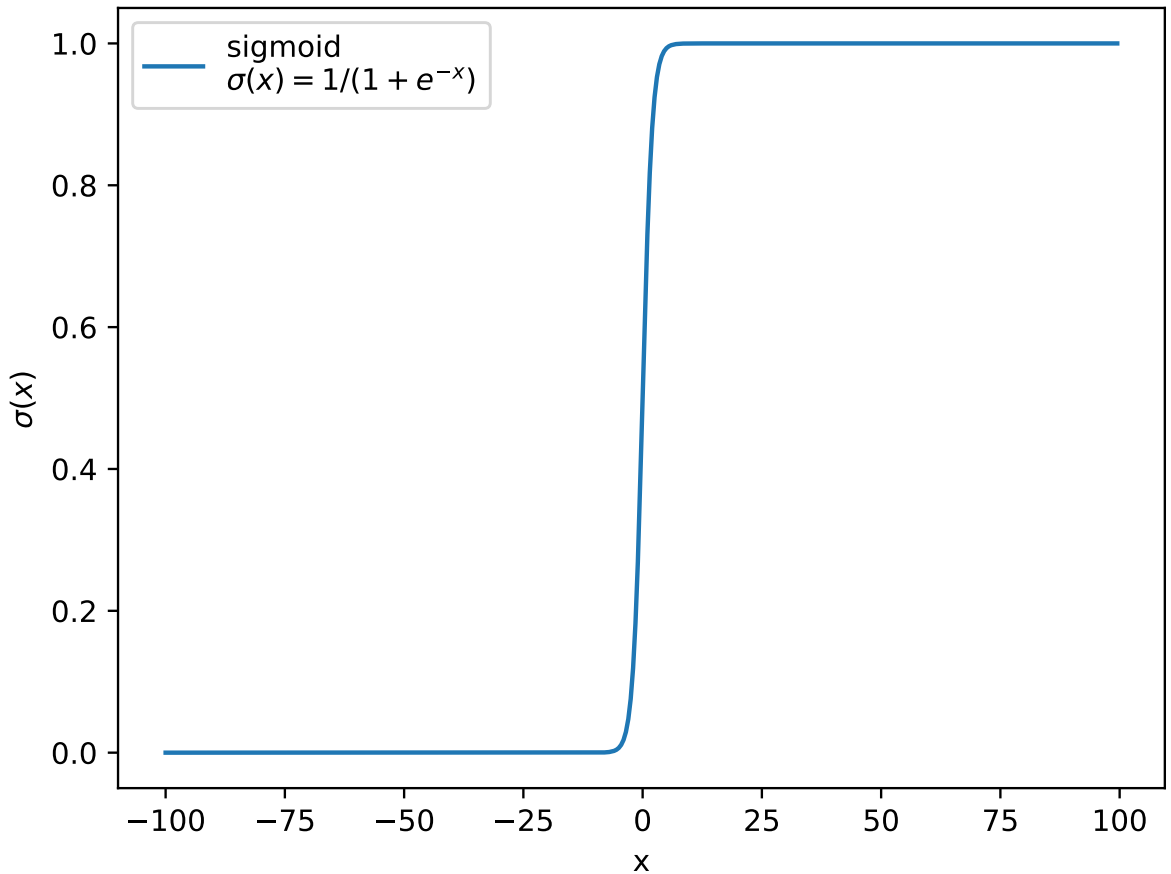
Sigmoid



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`

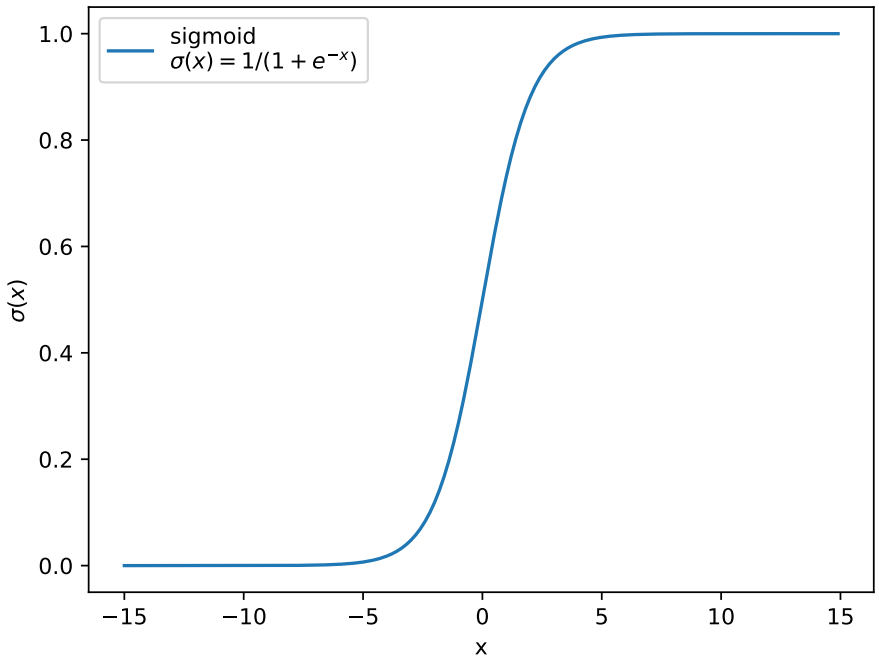
Non-convexity

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 2 :

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)

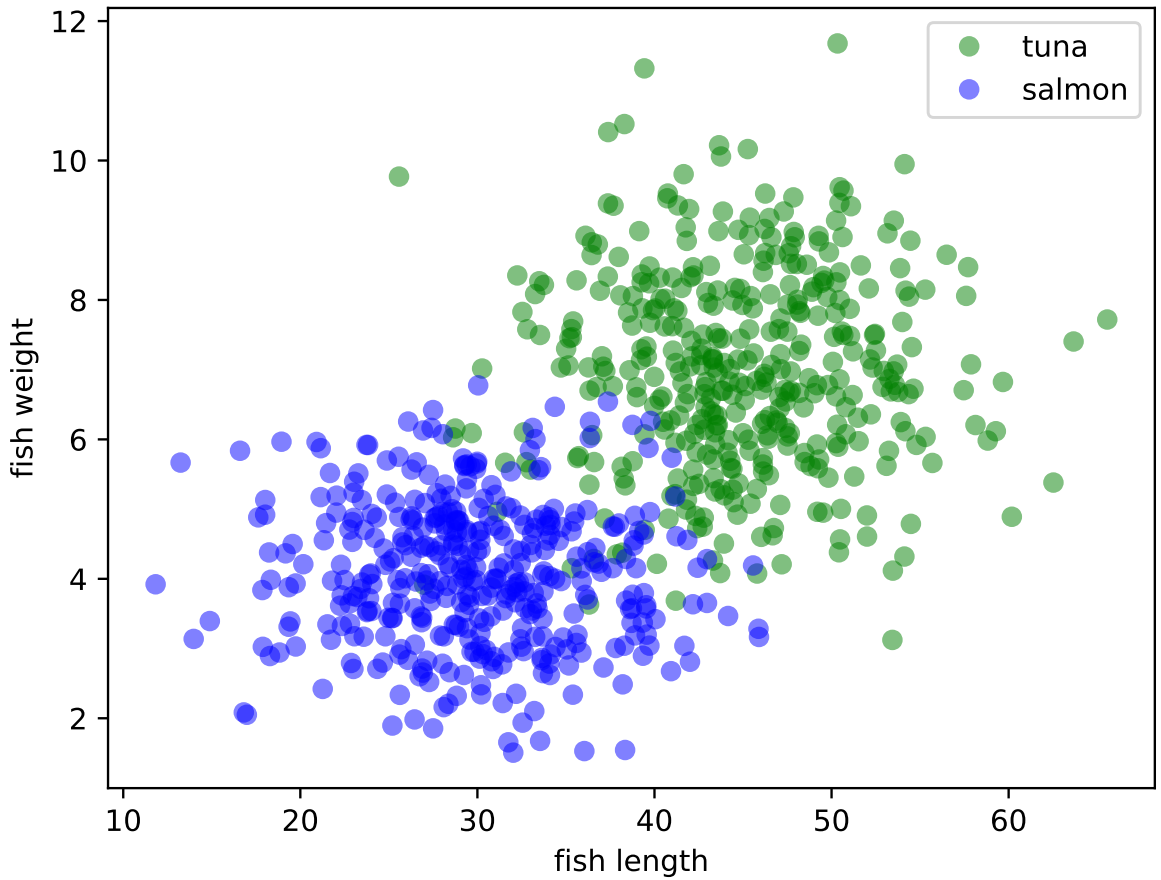
Optimization / regularization :

- dropout
- batch normalisation

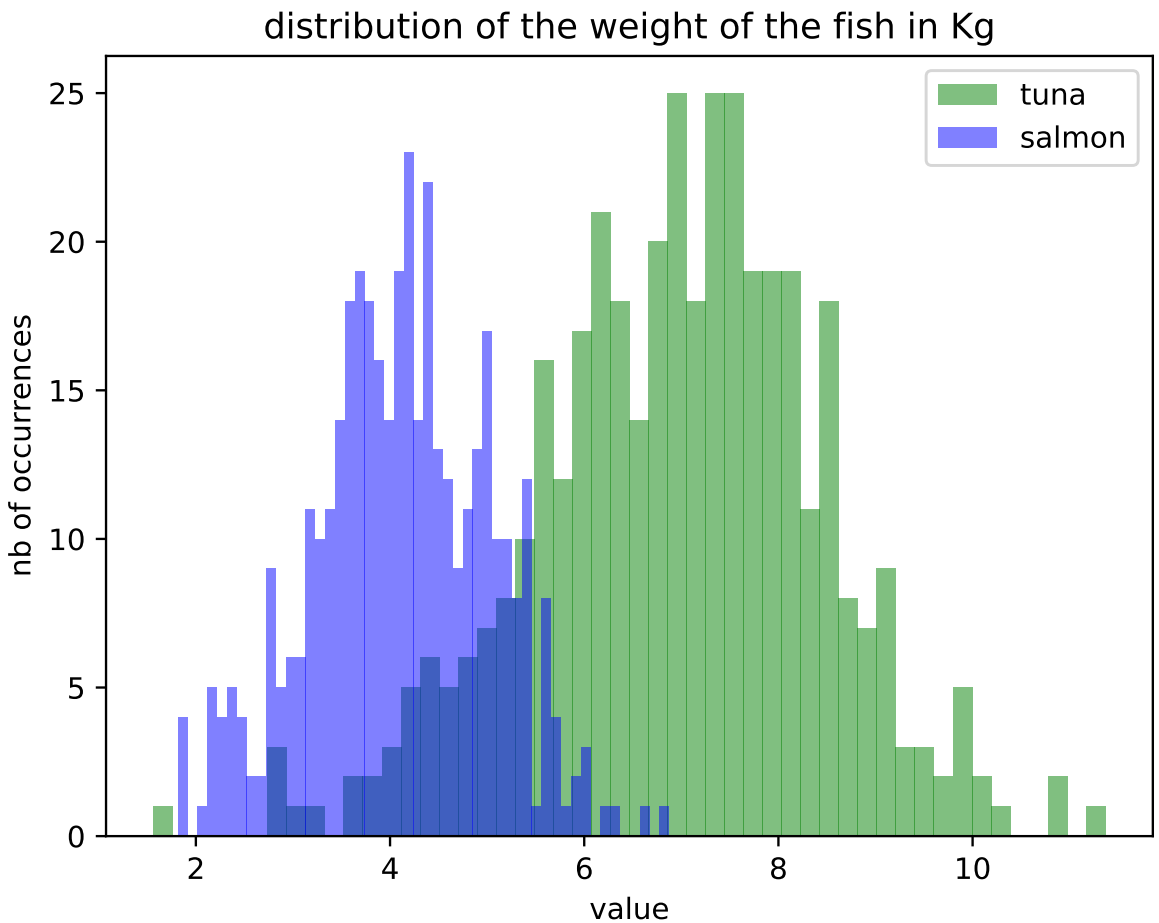
CART

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

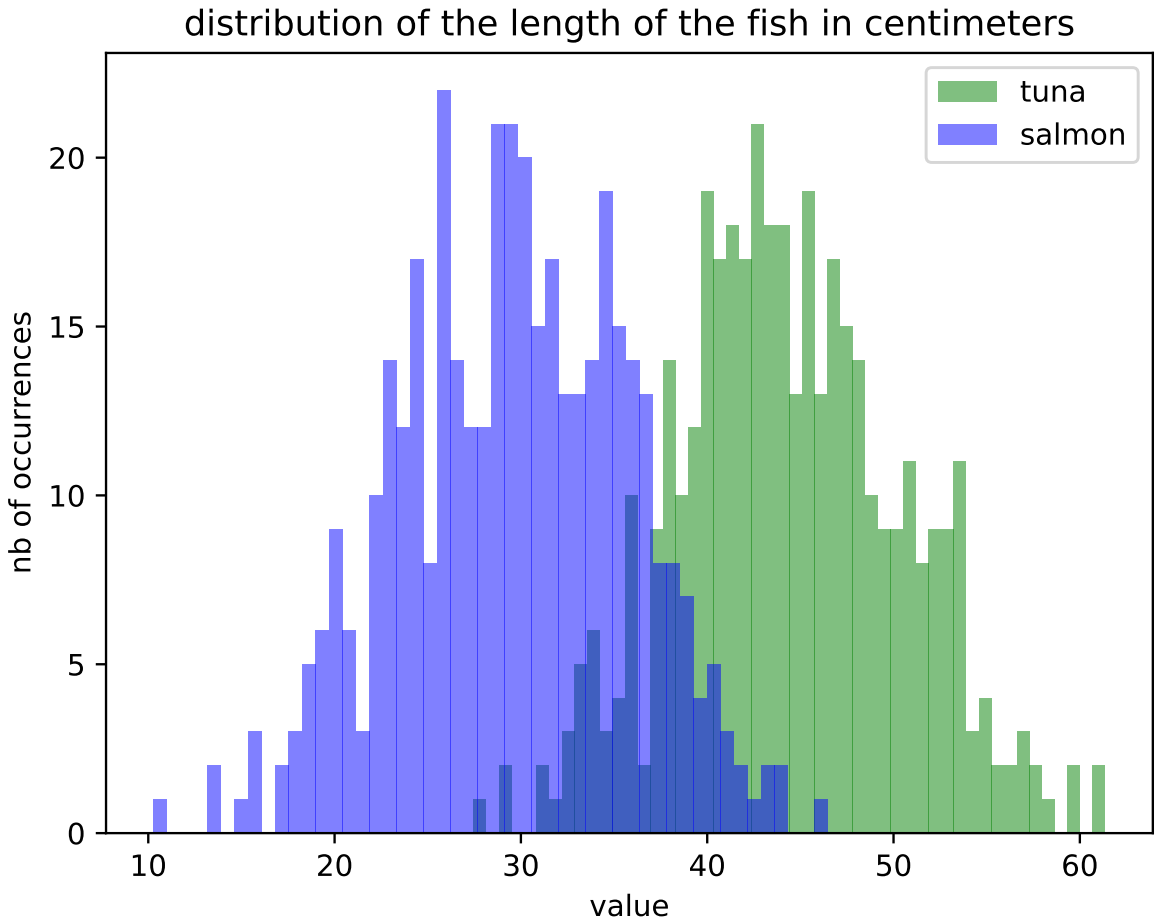
Example dataset : fishes



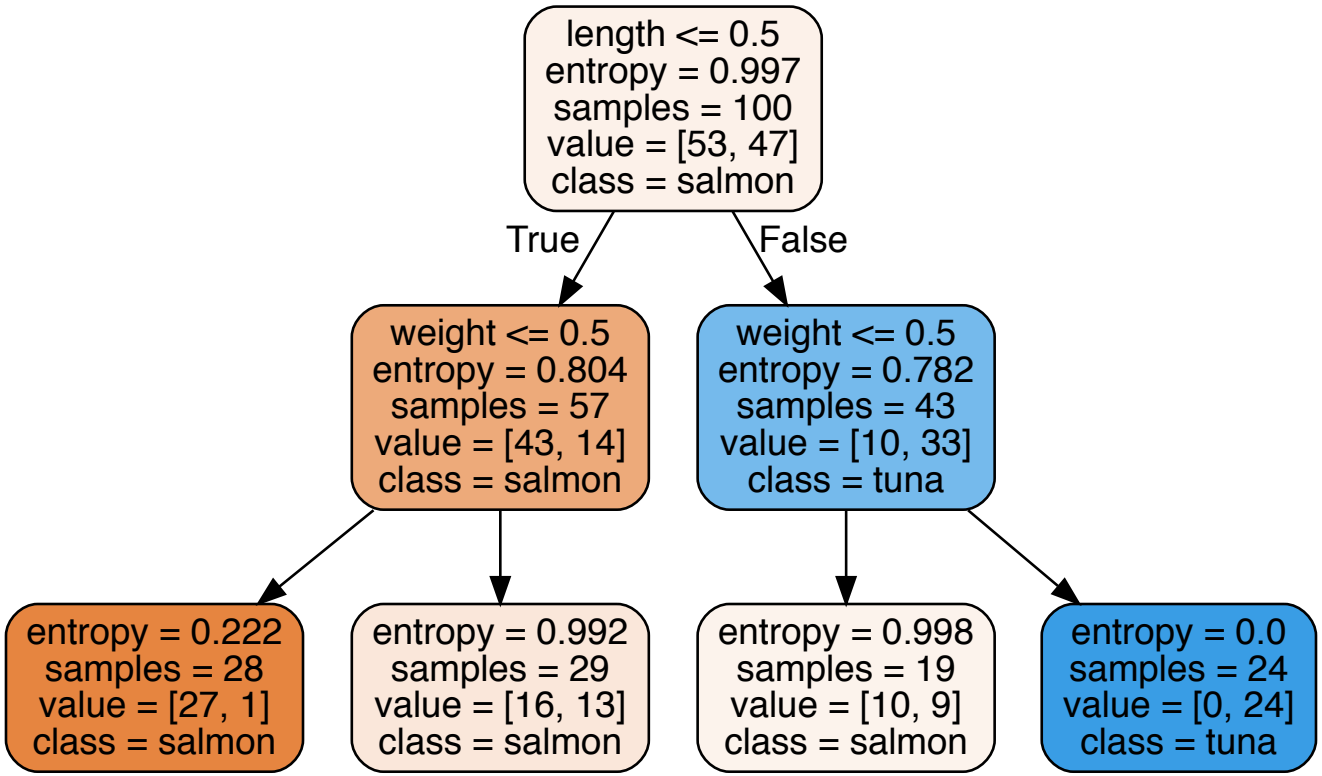
Fish dataset



Fish dataset

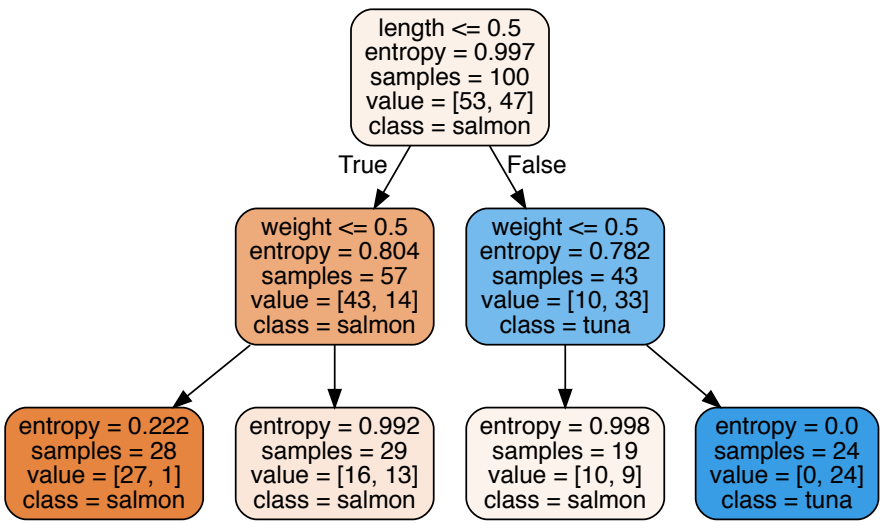


Example tree



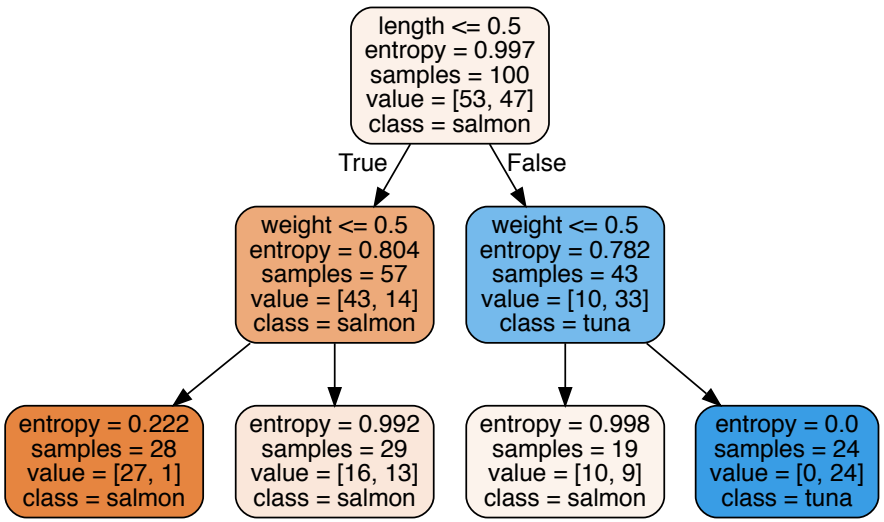
Splitting nodes

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

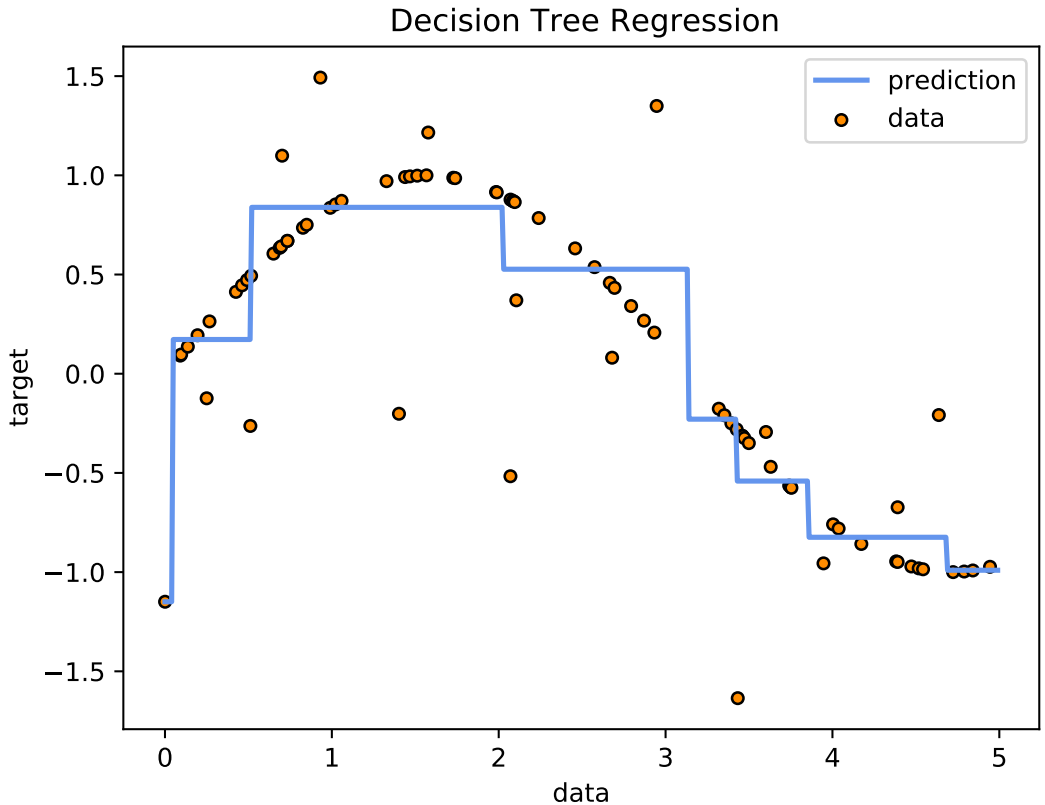


Splitting nodes

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

$$H(L) + H(R) \tag{13}$$

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 :

$$H(n) = \frac{1}{|n|} \sum_{i \in n} (y_i - \bar{y}_n)^2 \tag{14}$$

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 :

$$H(n) = -\sum_{l=1}^{L} p_n^l \log(p_n^l) \tag{15}$$

Exercice 3 : What are the maximum and minimum values of the entropy ?

Homogeneity criterion for classification : Gini impurity

The Gini impurity writes :

$$H(n) = \sum_{l=1}^{L} p_n^l (1 - p_n^l) \tag{16}$$

Homogeneity criterion for classification : Gini impurity

The Gini impurity writes :

$$H(n) = \sum_{l=1}^{L} p_n^l (1 - p_n^l) \tag{17}$$

Exercice 4 : 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

$$H(n) = 1 - \max_{l}(p_n^l) \tag{18}$$

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

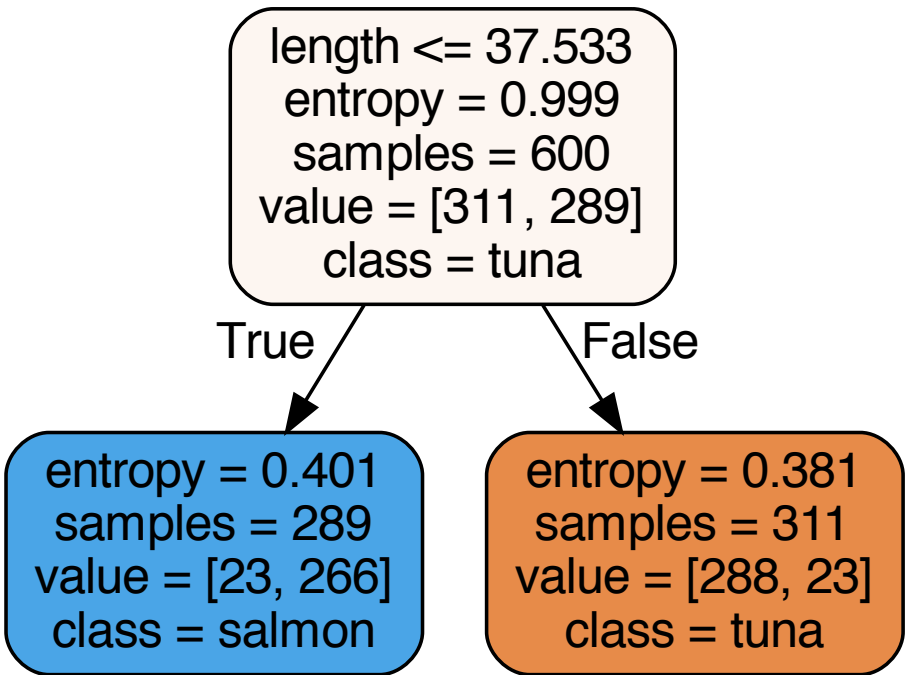
- If the segmentation is not restricted, the tree can fit the training dataset perfectly (overfitting).
- learning would then be highly dependent on the choice of the training dataset, leading to **instability.**
- To avoid it, **pruning** (élaguage) is used : it consists in restricting the size of the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

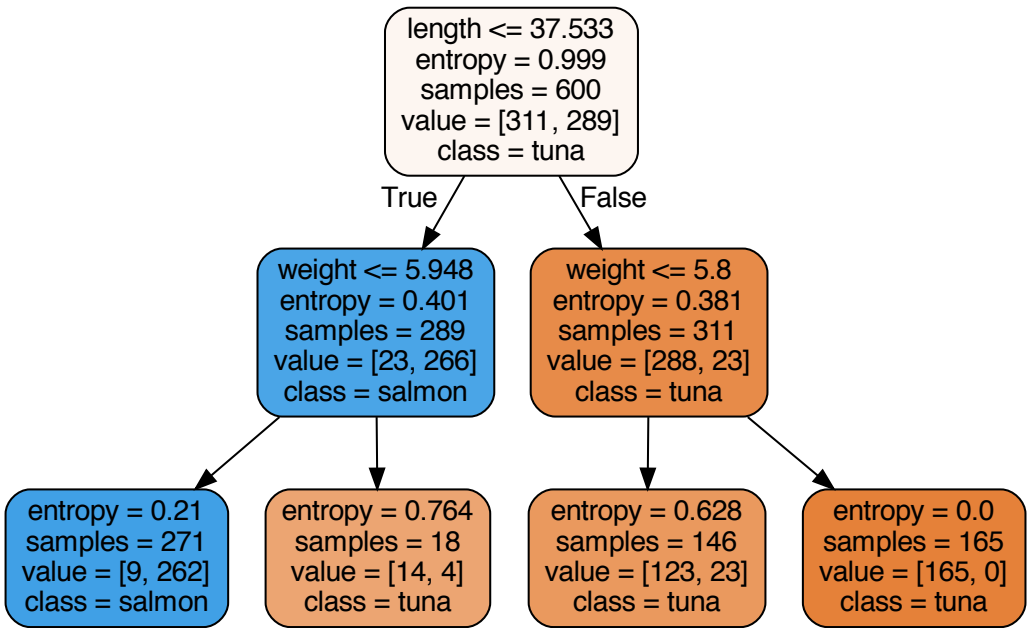
Pre-pruning by restricting divisions

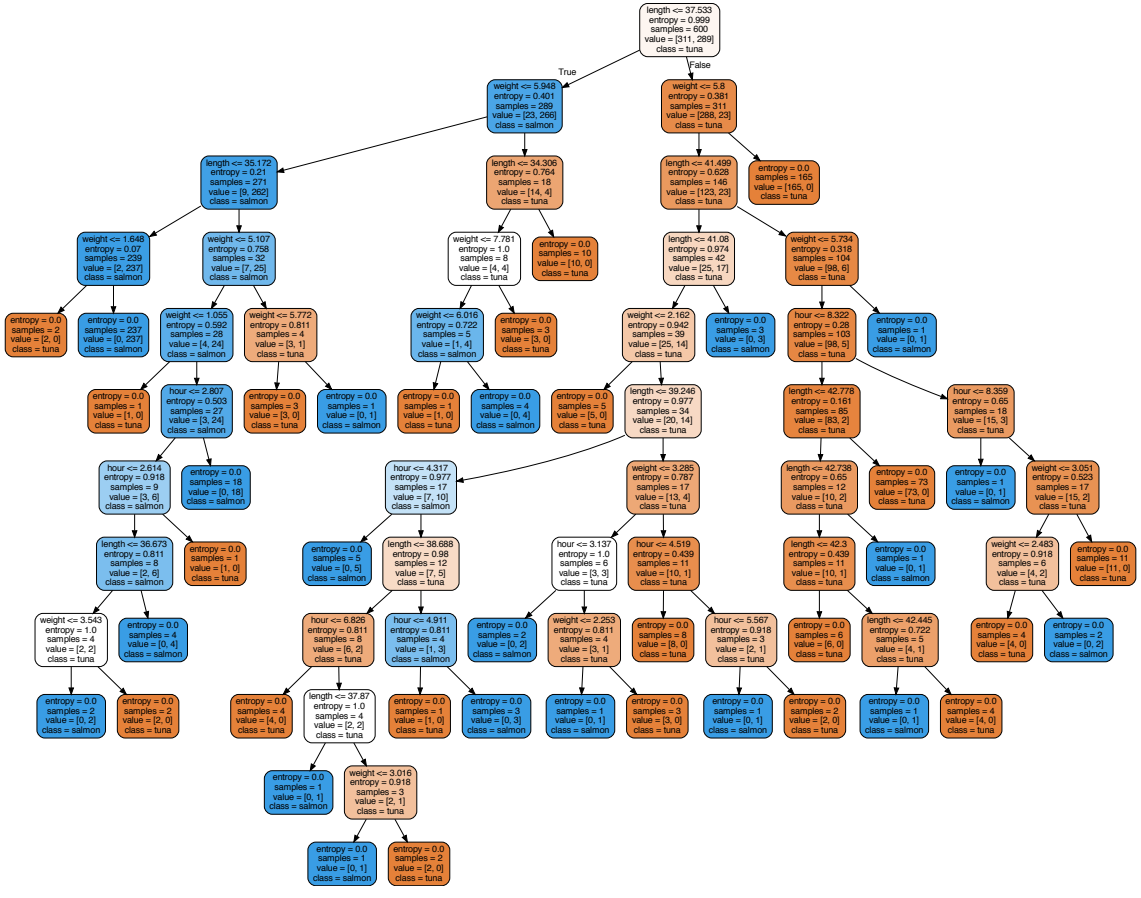
We can impose

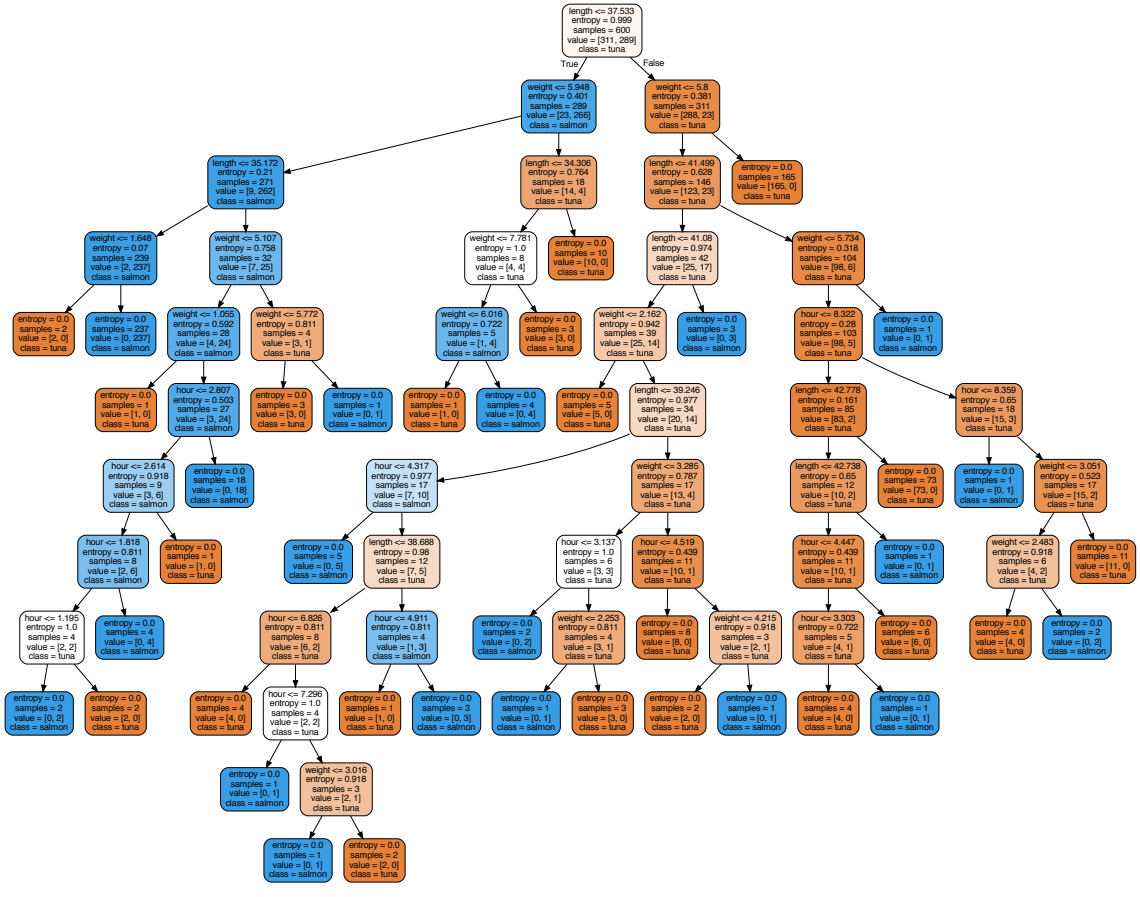
- minimum impurity decrease
- a minimum number of samples in a leaf node
- a minimum number of samples to authorize splitting a node

Simulation : fish dataset, max depth=1

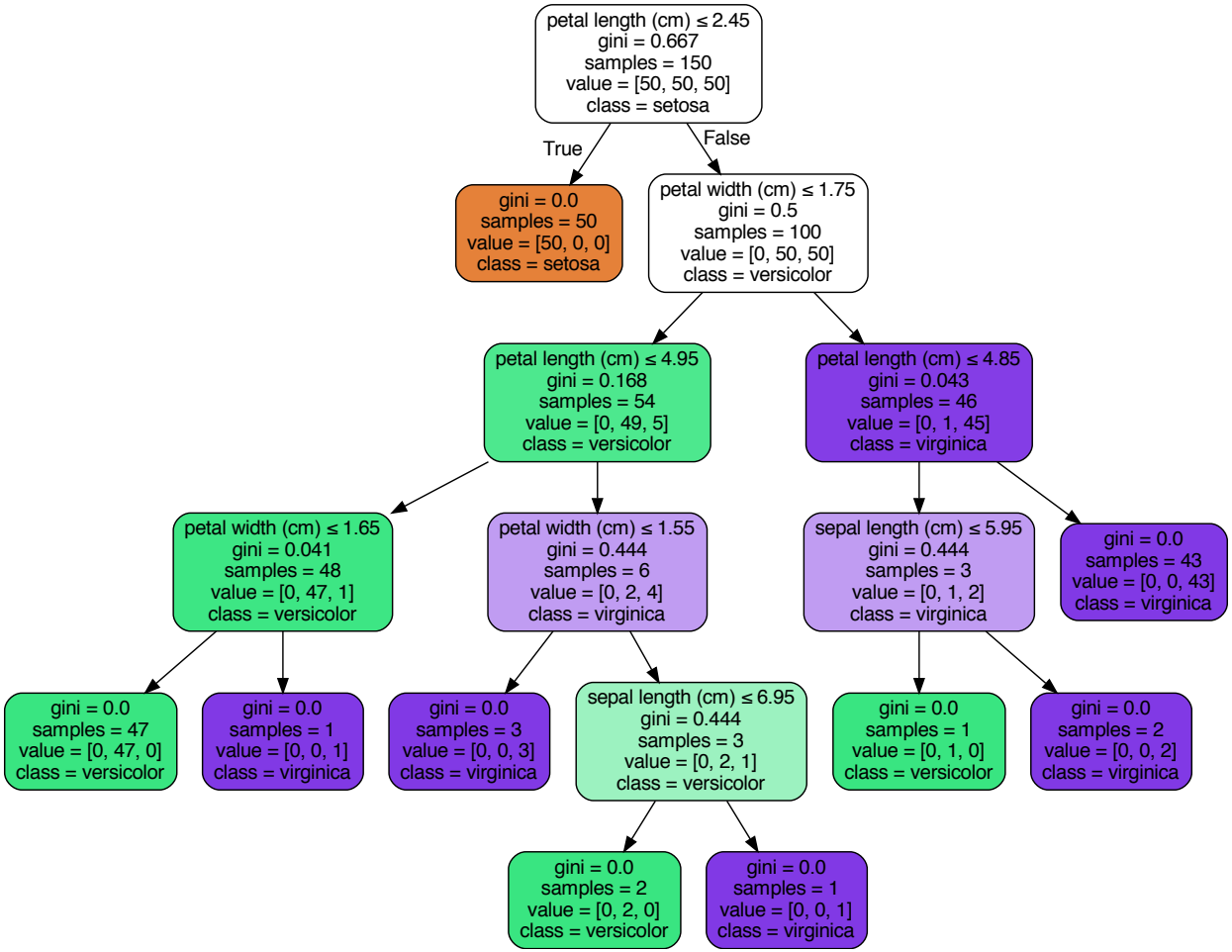



length <= 37.533
entropy = 0.999
samples = 600
value = [311, 289]
class = tuna
True
False
weight <= 5.948
entropy = 0.401
samples = 289
value = [23, 266]
class = salmon
weight <= 5.8
entropy = 0.381
samples = 311
value = [288, 23]
class = tuna
entropy = 0.21
samples = 271
value = [9, 262]
class = salmon
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

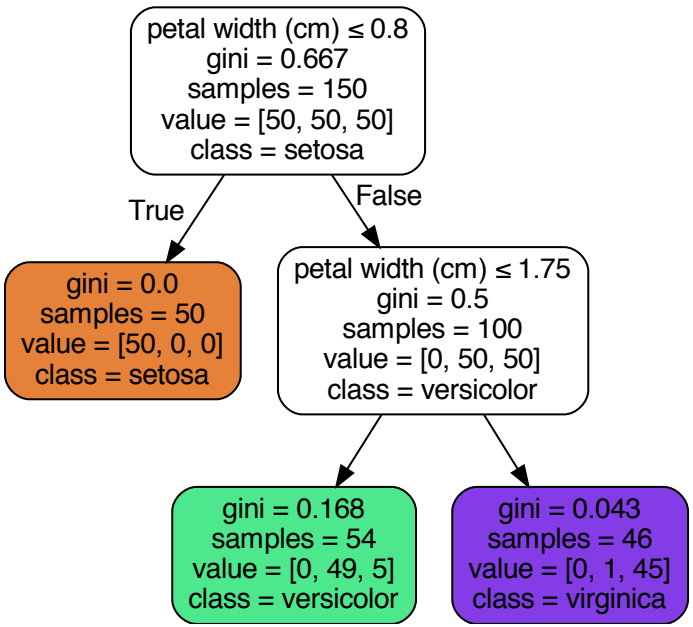


Figure – Avoid overfitting with pre-pruning. In this case, impose a minimum impurity decrease of 0.1

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.

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