EE-311—Apprentissage et intelligence artificielle

7. Arbres de décision et méthodes ensemblistes

François Fleuret & Michael Liebling

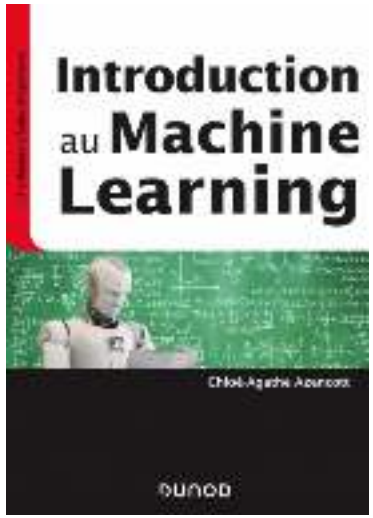
https://moodle.epfl.ch/course/view.php?id=16090

4 avril 2024 (compilé le 3 avril 2025)

idiap RESEARCH INSTITUTE

EPFL

Ouvrage de référence et source



Ces transparents sont basés en grande partie sur le texte de Chloé-Agathe Azencott “Introduction au Machine Learning”, Dunod, 2019
ISBN 978-210-080153-4

L’auteure a mis le texte (sans les exercices) à disposition ici : http://cazencott.info/dotclear/public/lectures/IntroML_Azencott.pdf

Avertissement : Bien que ces transparents partagent la notation mathématique, la structure de l’exposition (en partie), et certains exemples avec le livre, ils ne constituent qu’un complément et non un remplacement ou une source unique pour la couverture des matières du cours. À ce titre, ces transparents ne se substituent pas au texte.

Arbres de décision et méthodes ensemblistes

- Arbres de décisions
  - Introduction : motivation et définitions
  - Algorithme CART : “Classification And Regression Tree”
- Méthodes ensemblistes (combinaison de prédicteurs)
  - méthodes parallèles : bagging
    - forêts par bootstrapping
    - forêts aléatoires
  - méthodes séquentielles : boosting

Les méthodes vues jusqu'ici reposent fortement sur la *structure Euclidienne* de l'espace : distance, produits scalaires.

Les techniques à noyaux permettent de profiter d'une métrique plus pertinente, mais dans certains cas cela n'est pas suffisant, et il est nécessaire de recourir à des méthodes plus souples, telles que les *arbres de décision*.

Les arbres de décisions en apprentissage automatique peuvent être vus comme une formalisation des stratégies pour le “jeu des 20 questions” :

Q : Est-ce un humain ?

R : Oui

Q : Est-elle/il encore vivant ?

R : Non

Q : Est-ce un homme ?

R : Oui

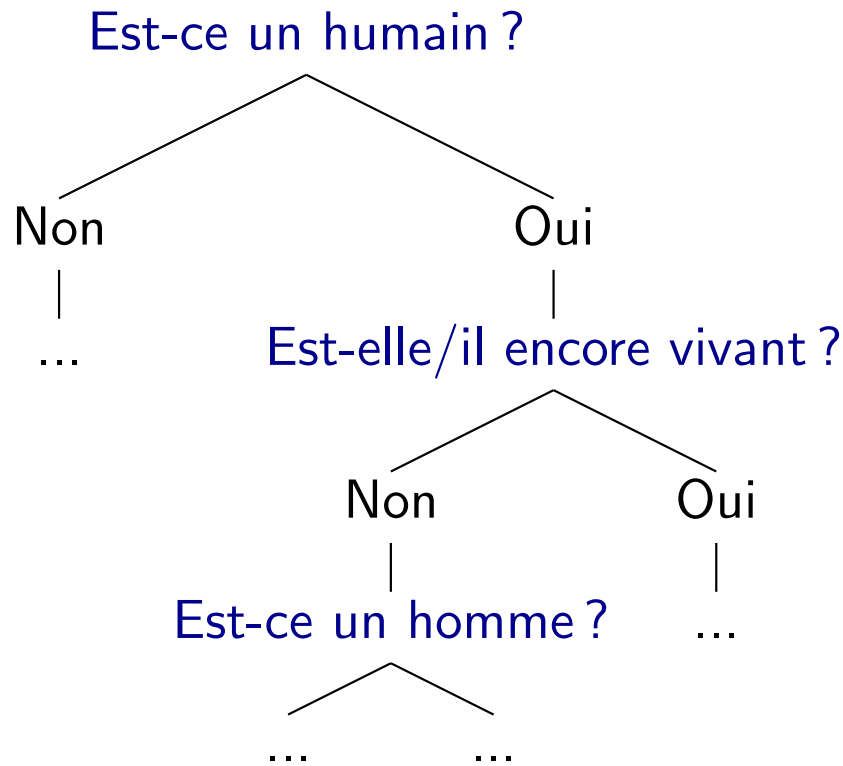
Q : Était-il européen ?

...

Note : voir par exemple https://fr.akinator.com/

La stratégie complète peut-être définie avant de commencer la partie, selon les statistiques que l'on connaît des choix des gens.

Elle peut être représentée à l'aide d'un arbre :



Pour la prédiction, un arbre de décision peut être vu comme une forme simple de “test adaptatif”, dans lequel la propriété à tester dépend de ce qui a déjà été testé et des réponses obtenues.

Pour la prédiction, un arbre de décision peut être vu comme une forme simple de “test adaptatif”, dans lequel la propriété à tester dépend de ce qui a déjà été testé et des réponses obtenues.

Pour l’apprentissage, comme dans le jeu des 20 questions, on essaye de poser la question “la plus efficace”, pour arriver à une décision le plus rapidement possible.

Bien que cette méthode soit simple et remonte aux années 80s, les arbres de décisions restent une méthode très performante pour des problème concrets d'apprentissage automatique.

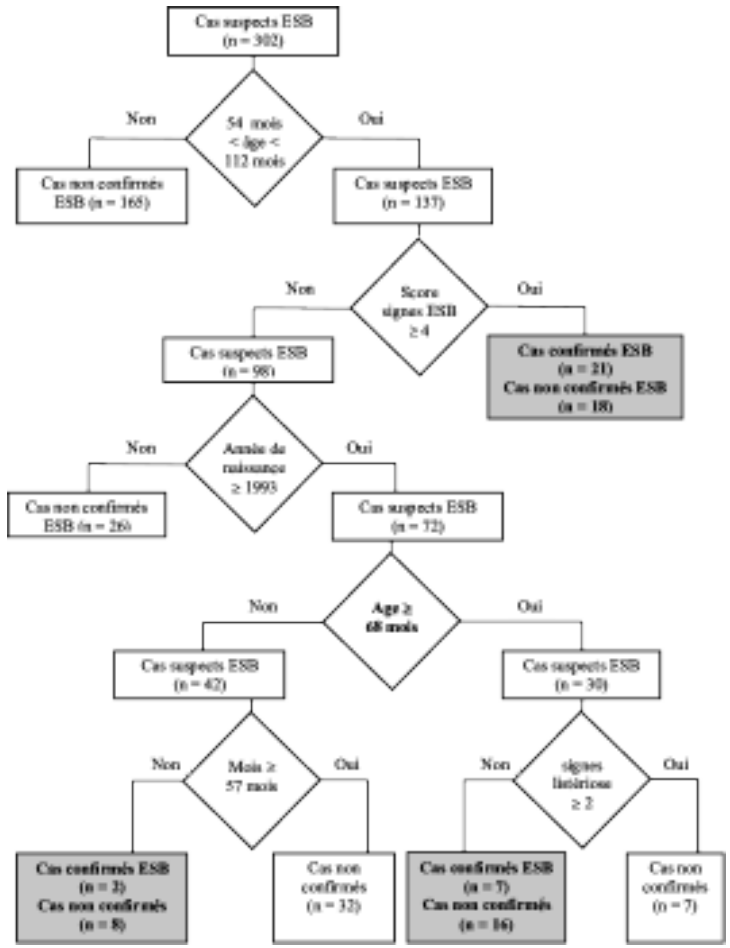
We evaluate 179 classifiers arising from 17 families (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, /. . ./

The random forest is clearly the best family of classifiers (3 out of 5 bests classifiers are RF), /. . ./

(Fernández-Delgado et al., 2014)

Exemple : arbre de décision pour diagnostic médical

Le concept d'arbre de décision est général et n'est pas limité à l'apprentissage automatique.



Source : C Saegerman, *et al.* "Amélioration de la détection d'une maladie émergente : exemple de l'encéphalopathie spongiforme bovine," Epidémiologie et Santé Animale, Vol. 44 pp. 61–77, 2003

Exemple : identification d'un fruit

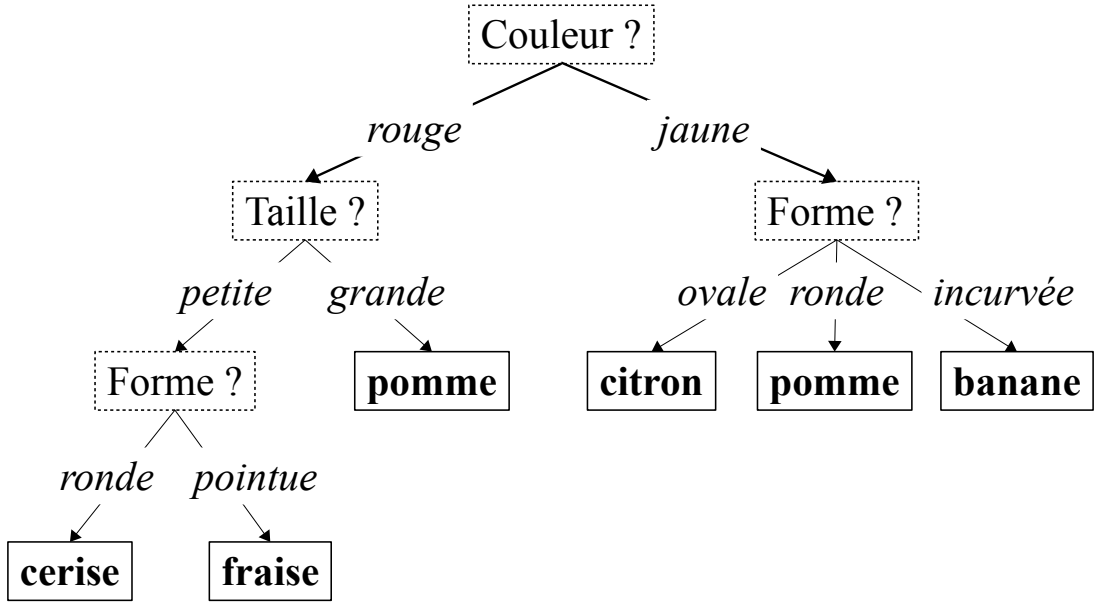


FIGURE 9.1 – Exemple d'arbre de décision pour étiqueter un fruit.

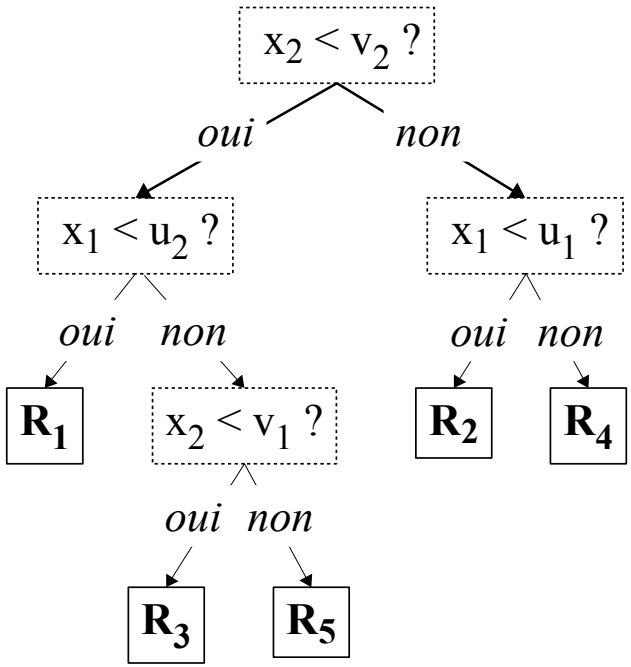
Azencott

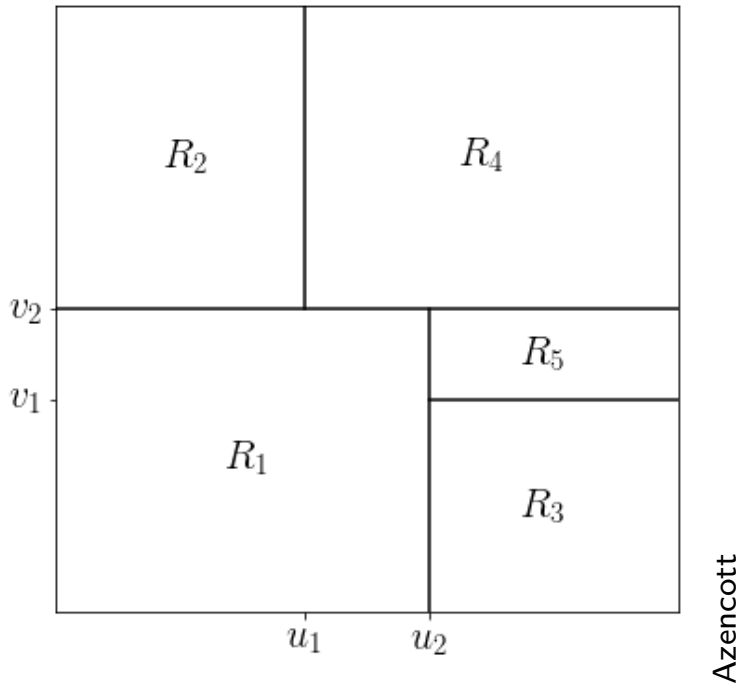
Cet exemple (non-binaire !) illustre plusieurs aspects des arbres de décision :

- ils peuvent traiter des attributs discrets ou valeurs continues, valeurs catégorielle
- ils peuvent faire une prédiction multi-classe,
- ils traitent naturellement des classes multi-modales (objets peuvent apparaître dans plusieurs catégories) .

Exemple avec interprétation géométrique

Méthode applicable aussi si on travaille dans un espace avec structure euclidienne

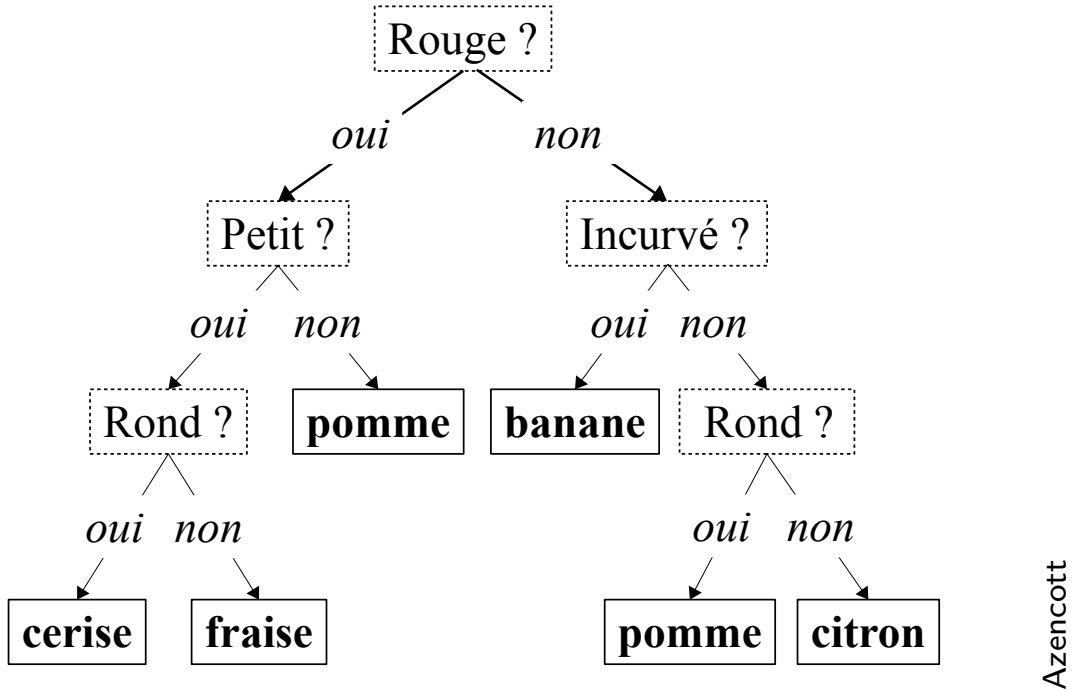




Azencott

Un arbre de décision partitionne récursivement l'espace des observations.

Transformation d'un arbre non-binaire en un arbre binaire (est toujours possible)



Pour la suite, et sans perte de généralité, on ne considérera que des arbres *binaires*, c'est à dire dont chaque nœud possède exactement deux enfants.

Algorithme CART ("Classification And Regression Tree")

Une méthode pour construire un arbre de décision

- publié par Leo Breiman *et al.* en 1984 [1]
- produit des arbres binaires
- critère de segmentation : indice de diversité de Gini.
- seule une propriété est considéré à chaque étape

1. Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. "Classification and Regression Trees," Chapman & Hall (Boca Raton), 1984

Définitions : arbre binaire, feuilles, noeuds, arbres enfants

Soit $\mathcal{X}$ l'espace des observations, et $\mathcal{Y}$ l'espace des valeurs à prédire (classe discrète ou vecteur de valeurs continues).

On peut définir un arbre binaire $\mathcal{T}$ (qui associe $\mathcal{X}$ et $\mathcal{Y}$) de manière récursive. Un arbre binaire est soit :

- Une feuille, qui porte une valeur $\hat{y} \in \mathcal{Y}$.

  La prédiction de $\mathcal{T}$ est alors $\forall x \in \mathcal{X}, \mathcal{T}(x) = \hat{y}$

- Un nœud, qui constitue la racine de l'arbre, et qui porte deux régions R_{L} et R_{R} avec $R_{\mathrm{L}} \cap R_{\mathrm{R}} = \emptyset$ et $R_{\mathrm{L}} \cup R_{\mathrm{R}} = \mathcal{X}$,
  et deux arbres enfants $\mathcal{T}_{\mathrm{L}}$ et $\mathcal{T}_{\mathrm{R}}$.

  La prédiction de $\mathcal{T}$ est alors

$$\forall x \in \mathcal{X},\ \mathcal{T}(x) = \begin{cases} \mathcal{T}_{\mathrm{L}}(x) & \text{si } x \in R_{\mathrm{L}} \\ \mathcal{T}_{\mathrm{R}}(x) & \text{si } x \in R_{\mathrm{R}}. \end{cases}$$

La *profondeur* d'un arbre (depth) est la distance maximale entre la racine et toute feuille.

Tests booléens considérés

Un des principaux avantages des arbres de décision est leur capacité à traiter des observations combinant des grandeurs discrètes / catégorielles et continues.

Bien que les tests placés aux nœuds puissent être arbitraires, nous considérerons dans la suite ceux de l'algorithme le plus classique : CART ("Classification And Regression Tree").

Tests booléens dans l'algorithme CART

Chaque nœud d'un arbre construit par CART ne considère qu'une coordonnée x_j de x, appelée "variable séparatrice".

Selon le type de cette variable x_j, les régions sont définie comme suit :

- Si x_j est booléenne, la région R_{L} est

$$R_{\mathrm{L}} = \{x \in \mathcal{X},\ x_j = 0\}.$$

- Si x_j est discrète la région R_{L} dépend de l'appartenance de x_j à un ensemble de valeurs $\mathcal{S}$:

$$R_{\mathrm{L}} = \{x \in \mathcal{X},\ x_j \in \mathcal{S}\}.$$

- Si x_j est continue R_{L} est définie par un seuil s :

$$R_{\mathrm{L}} = \{x \in \mathcal{X},\ x_j < s\}.$$

Motivation : comment choisir quoi mettre aux noeuds, quand s'arrêter et mettre des feuilles ?

Les algorithmes d'apprentissage classiques (tel que CART) pour construire un arbre de décision sont gloutons et récursifs.

Chaque nœud est construit selon les données d'apprentissage qui y parviennent. Il dépend donc des nœuds qui ont été construits au-dessus, et l'arbre résultant n'est globalement pas optimal, en général.

Critère pour décider si un noeud sera une feuille

Étant donné un ensemble d'apprentissage à la position d'un noeud :

$$(x_n, y_n)_{n=1,\ldots,N} :$$

1. Si un critère d'arrêt est vérifié, par exemple $N \leq N_{\text{min}}$, alors le nœud sera une feuille, avec, comme valeur à prédire pour cette feuille :

- pour la régression $\hat{y} = \frac{1}{N} \sum_n y_n$ (= la moyenne),
- pour la classification $\hat{y} = \operatorname{argmax}_c \underbrace{\sum_n \delta(c, y_n)}_{\text{nb d'exemples de classe c}}$

  (= la classe majoritaire).

Si le noeud n'est pas une feuille

2. Dans le cas où le nœud n'est pas une feuille, on doit sélectionner :

- une variables séparatrice x_j,
- soit
  - une valeur pour s (si variable continue) ou
  - $\mathcal{S}$ (si variable discrète).

Ce choix est fait pour optimiser un critère qui estime la facilité à faire la prédiction dans les arbres fils.

Régression : critère pour déterminer la qualité du choix de la variable séparatrice et du seuil

Pour la régression, un critère naturel pour déterminer la qualité d'un seuil s, est la *variance empirique conditionnelle*.

Pour rappel, les régions R_{L} et R_{R} sont définies par le seuil s :

$$R_{\mathrm{L}} = \{x \in \mathcal{X},\ x_j < s\}, \quad R_{\mathrm{R}} = \{x \in \mathcal{X},\ x_j \geq s\}.$$

Si $\hat{y}_{\mathrm{L}}$ et $\hat{y}_{\mathrm{R}}$ sont les deux prédictions qui seraient faites dans les arbres fils si ces derniers étaient des feuilles, le critère est défini comme :

$$\sum_{n:\, x_n \in R_{\mathrm{L}}} (y_n - \hat{y}_{\mathrm{L}})^2 + \sum_{n:\, x_n \in R_{\mathrm{R}}} (y_n - \hat{y}_{\mathrm{R}})^2 .$$

(Mauvais) critère pour le cas de la classification

Pour la classification, chercher à simplement réduire l'erreur empirique est une mauvaise stratégie.

Considérons un problème à quatre classes, avec 100 exemples de chaque classe, et deux splits possibles :

	$\mathcal{T}_\text{L}$	$\mathcal{T}_\text{R}$	Erreur
Split #1	$\begin{bmatrix} ✓ & ✗ & ✗ & ✗ \\ 100 & 99 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} ✗ & ✗ & ✗ & ✓ \\ 0 & 1 & 99 & 100 \end{bmatrix}$	50%
Split #2	$\begin{bmatrix} ✓ & ✗ & ✗ & ✗ \\ 100 & 50 & 50 & 0 \end{bmatrix}$	$\begin{bmatrix} ✗ & ✗ & ✗ & ✓ \\ 0 & 50 & 50 & 100 \end{bmatrix}$	50%

Le split #1 est meilleur, puisqu'il suffira de résoudre des problèmes à deux classes dans les nœuds suivants.

Pourtant les deux splits ont le même taux d'erreur dans les feuilles qu'ils créent.

$\Rightarrow$ on aimerait un critère qui tient compte des distributions créées !

Critère pour le cas de la classification

On préfère donc utiliser un critère d'impureté qui estime de manière plus pertinente la qualité d'un split.

Les deux grandeurs classiques sont

1. l'entropie de Shanon et
2. l'impureté de Gini.

Les deux sont définies pour une distribution et somment une fonction concave des probabilités. Elles ne dépendent donc évidemment pas de l'ordre, et sont d'autant plus petites que les probabilités sont concentrées sur quelques classes.

Entropie de Shannon

Étant donnée une distribution de probabilités $p_1, \ldots, p_C$, l'entropie de Shannon est égale à :

$$\mathbb{H}(p) = -\sum_{c=1}^{C} p_c \log p_c$$

avec la convention que $0 \log 0 = 0$.

Les cas extêmes :

Si la distribution est déterministe, donc si tous les p_c sont nuls sauf une égal à 1, $\mathbb{H}(p) = 0$.

Si la distribution est uniforme, $p_c = 1/C$, alors $\mathbb{H}(p) = \log C$.

Impureté de Gini

Étant donnée une distribution de probabilités $p_1, \ldots, p_C$, l'impureté de Gini est égale à :

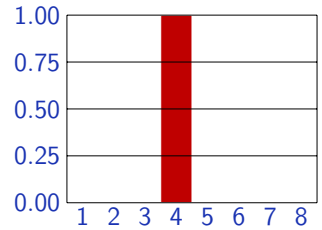
$$\text{Gini}(p) = \sum_{k=1}^{C} p_k(1 - p_k)$$

Les cas extêmes :

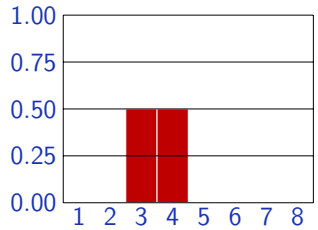
Si la distribution est déterministe, donc si tous les p_c sont nuls sauf une égal à 1, $\text{Gini}(p) = 0$.

Si la distribution est uniforme, alors $\text{Gini}(p) = 1 - 1/C$.

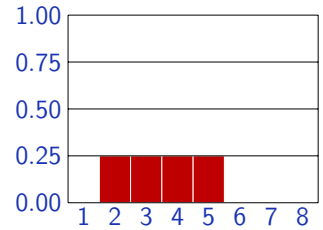
Illustration des critères sur différentes distributions



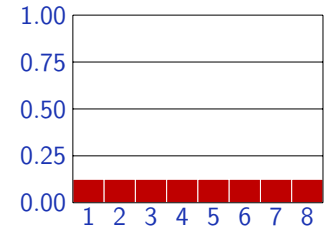
Error	$\mathbb{H}$	Gini
0.00	0.00	0.00



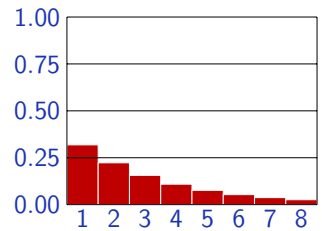
Error	$\mathbb{H}$	Gini
0.50	0.69	0.50



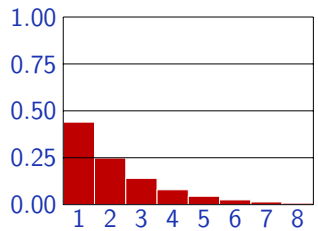
Error	$\mathbb{H}$	Gini
0.75	1.39	0.75



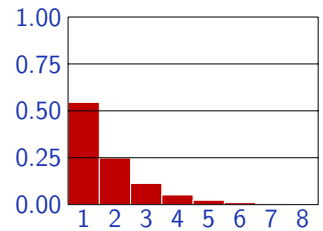
Error	$\mathbb{H}$	Gini
0.88	2.08	0.88



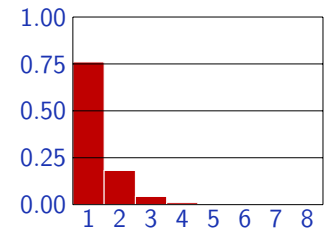
Error	$\mathbb{H}$	Gini
0.68	1.80	0.80



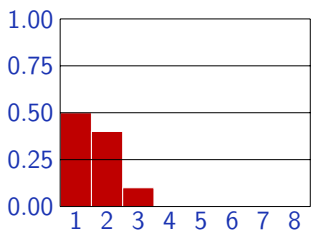
Error	$\mathbb{H}$	Gini
0.56	1.51	0.72



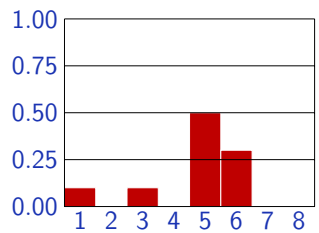
Error	$\mathbb{H}$	Gini
0.45	1.25	0.62



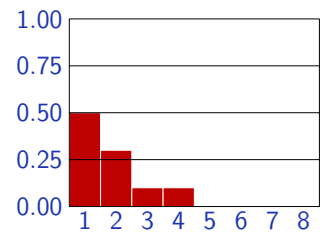
Error	$\mathbb{H}$	Gini
0.24	0.72	0.39



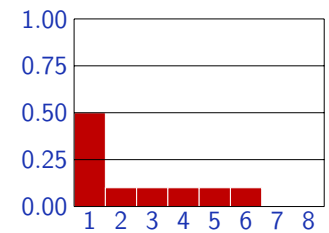
Error	$\mathbb{H}$	Gini
0.50	0.94	0.58



Error	$\mathbb{H}$	Gini
0.50	1.17	0.64



Error	$\mathbb{H}$	Gini
0.50	1.17	0.64



Error	$\mathbb{H}$	Gini
0.50	1.50	0.70

Note : les valeurs des deux critères ne dépendent pas de l'ordre des classes !

Caractérisation de la qualité d'un split sur la base de l'impureté

Pour caractériser la qualité d'un split :
1. Choisir une mesure d'impureté
2. Déterminer les 2 distributions empiriques engendrées par ce split
3. Calculer la moyenne pondérée des impuretés de ces 2 distributions :

$$|\{n : x_n \in R_{\mathrm{L}}\}| \operatorname{Imp}(p_{\mathrm{L}}) + |\{n : x_n \in R_{\mathrm{R}}\}| \operatorname{Imp}(p_{\mathrm{R}})$$

où Imp est $\mathbb{H}$ ou Gini et $p_{\mathrm{L}}, p_{\mathrm{R}}$ sont les distributions empiriques :

$$p_{\mathrm{L},c} = \frac{|\{n : x_n \in R_{\mathrm{L}}, y_n = c\}|}{|\{n : x_n \in R_{\mathrm{L}}\}|} \quad p_{\mathrm{R},c} = \frac{|\{n : x_n \in R_{\mathrm{R}}, y_n = c\}|}{|\{n : x_n \in R_{\mathrm{R}}\}|}$$

Interprétation : on reconnaît le calcul d'une moyenne de l'impureté à chacune des deux feuilles produites (basée sur des distributions *empiriques*), avec comme pondération le nombre d'éléments assignés à la feuille correspondante.

Algorithme CART : choix des variables séparatrices et des seuils ou ensembles

L'algorithme CART teste exhaustivement toutes les valeurs possibles $j = 1, \ldots, D$ pour la variable séparatrice et,

- pour les variables continues, étant donné un tri σ croissant

$$x_j^{\sigma(1)} < x_j^{\sigma(2)} < \cdots < x_j^{\sigma(N)},$$

il considère tous les $N - 1$ seuils s possibles :

$$s \in \left\{ \frac{x_j^{\sigma(1)} + x_j^{\sigma(2)}}{2}, \ldots, \frac{x_j^{\sigma(N-1)} + x_j^{\sigma(N)}}{2} \right\}.$$

- pour les variables discrètes il considère tous les partitionnements possibles pour $\mathcal{S}$.

$\Rightarrow$ la meilleure variable séparatrice et seuil ou ensemble correspondant sont choisis selon le critère du slide précédent.

Critère d'arrêt ou d'élagage de l'arbre

L'algorithme CART peut également limiter la complexité des arbres en optimisant une fonction de coût globale de la forme

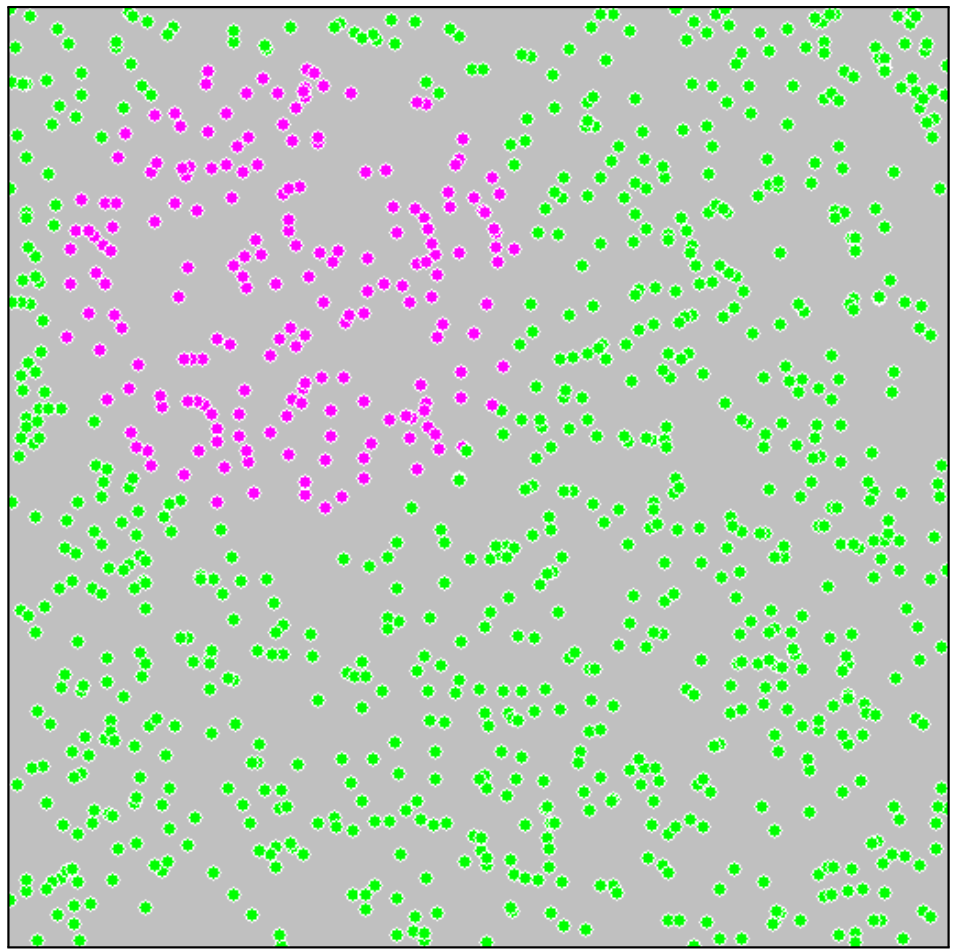
$$C_\lambda(\mathscr{T}) = \sum_{\mathbf{u}=1}^{|\mathscr{T}|} N_{\mathbf{u}} \, \mathrm{Imp}(p_{\mathbf{u}}) + \lambda |\mathscr{T}|$$

pour la classification, où $|\mathscr{T}|$ est le nombre de feuilles de l'arbre, et pour toute feuille $\mathbf{u}$, $N_{\mathbf{u}}$ le nombre d'exemples d'apprentissage qui y arrivent et $p_{\mathbf{u}}$ la distribution empirique de leurs classes.

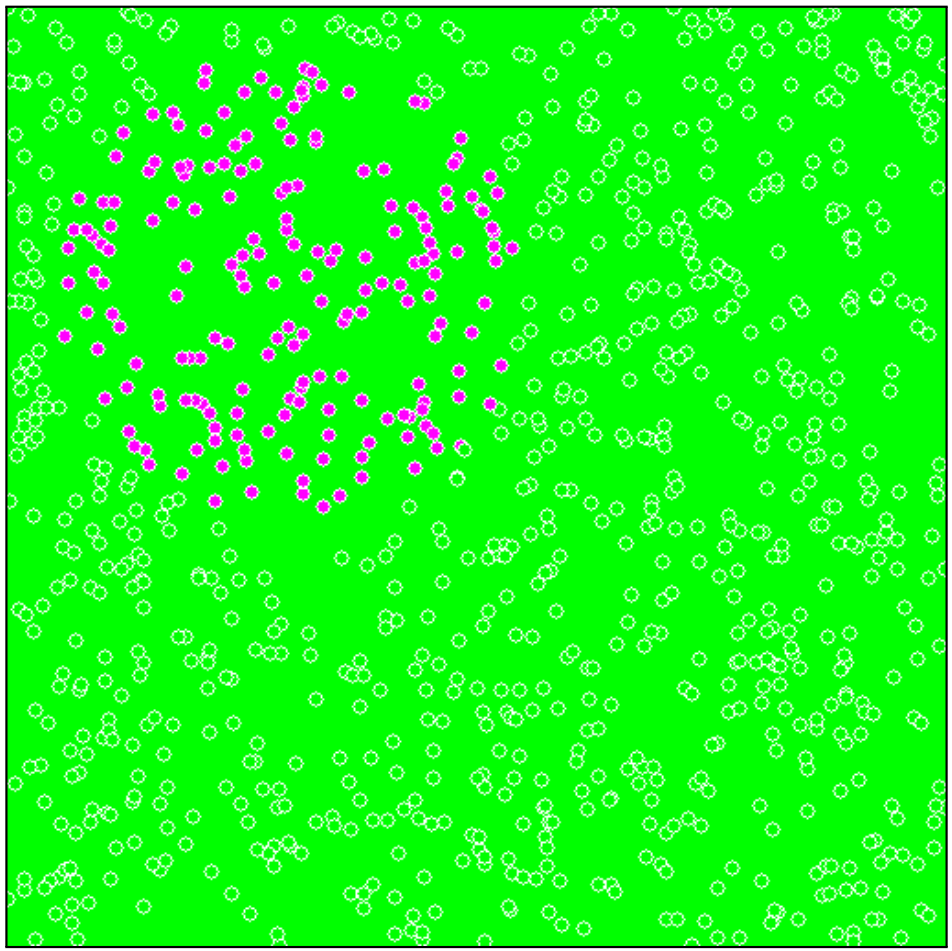
Une mesure similaire pour la régression repose sur l'erreur quadratique en prédiction.

Exemple d'apprentissage CART

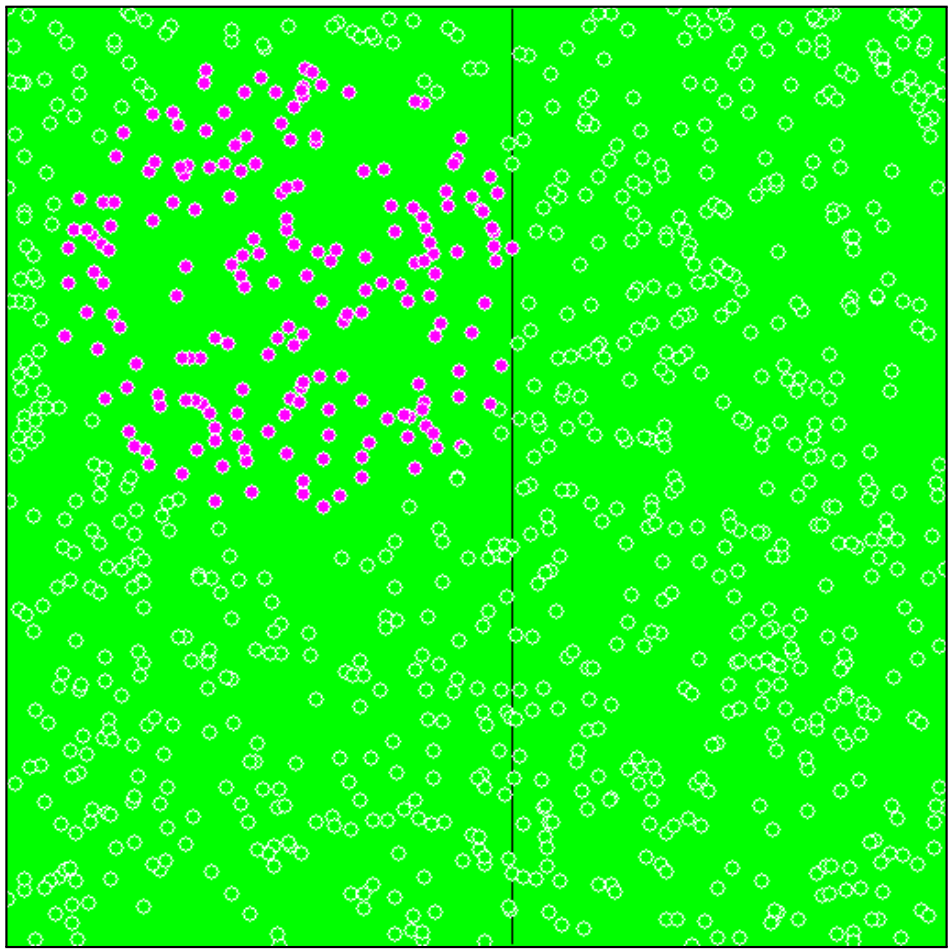
Nous pouvons illustrer l'apprentissage d'un arbre sur un exemple à deux classes (points verts ou magenta), dimension $D = 2$ et $N = 1,000$ exemples.



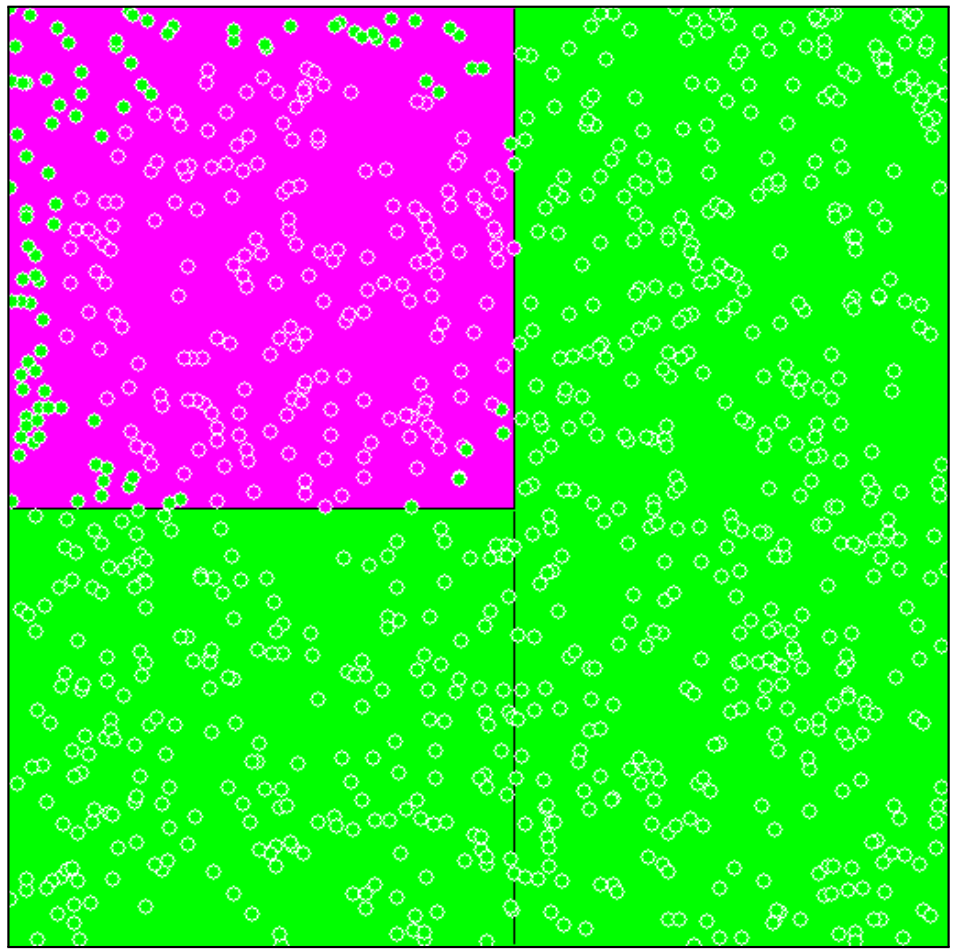
Profondeur : 0 (1 feuille ; couleur du fond = prédiction)



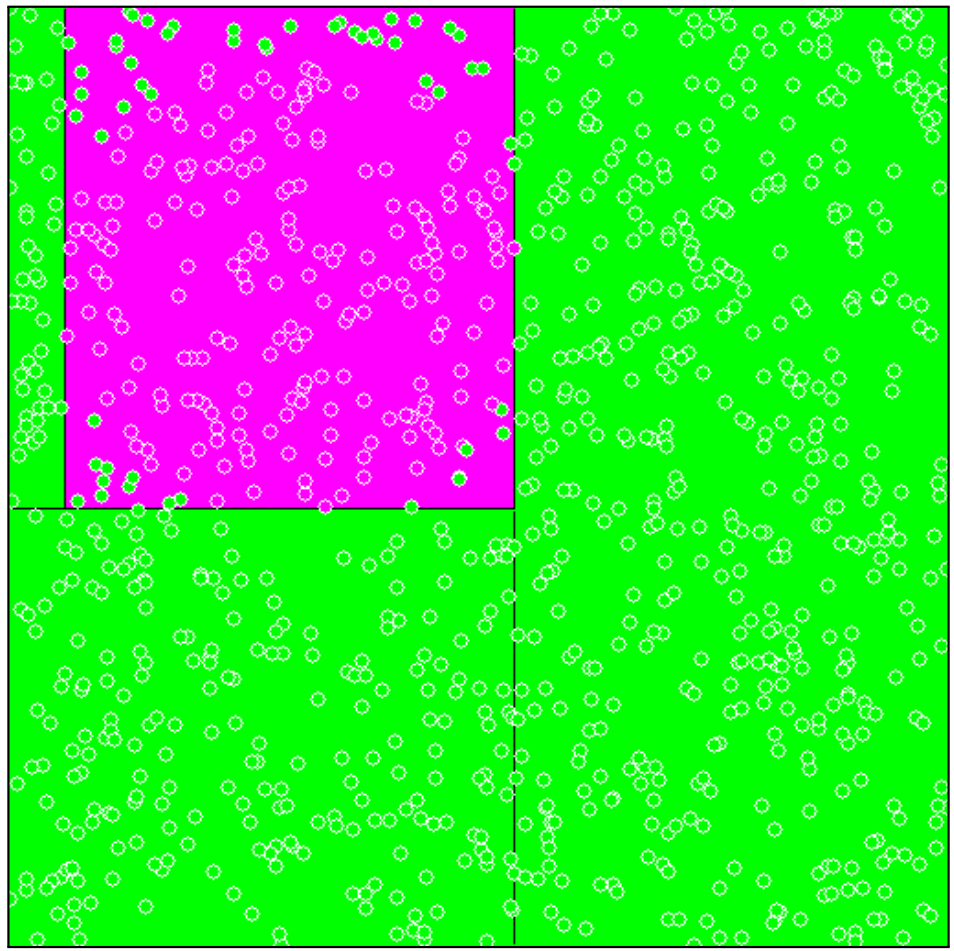
Profondeur : 1 (1 nouveau split, 2 feuilles au total)



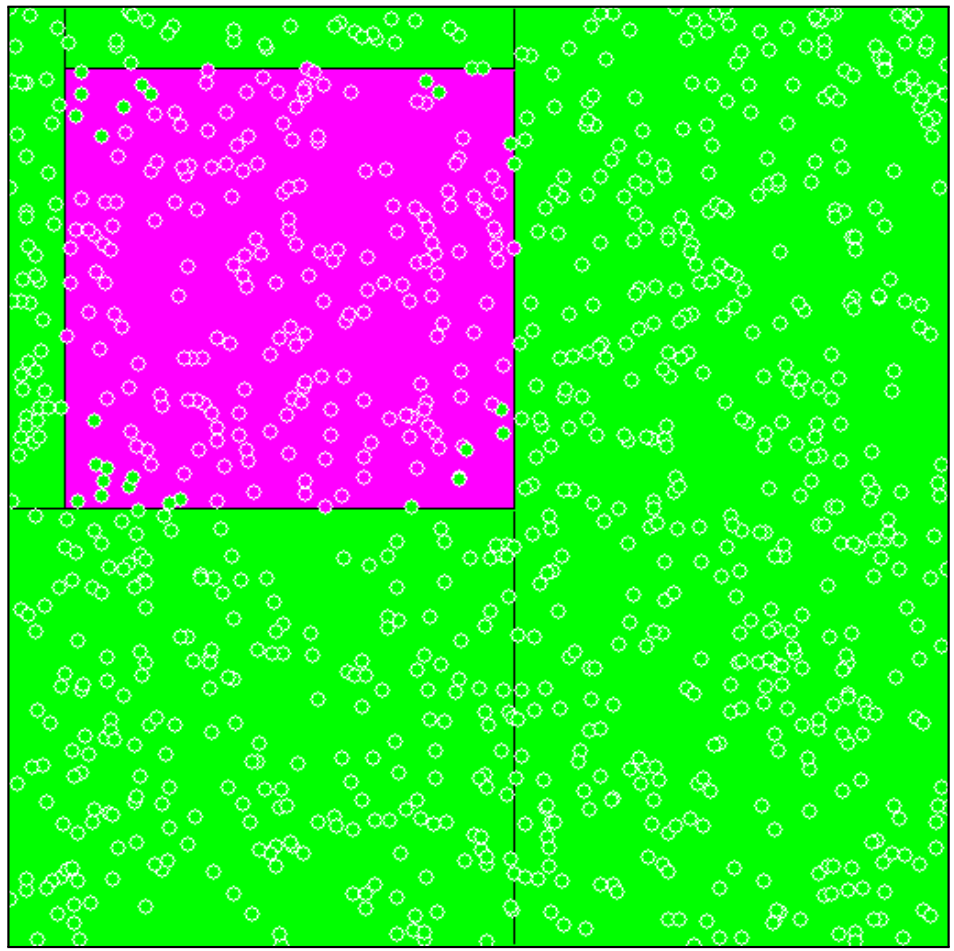
Profondeur : 2 (1 nouveau split, 3 feuilles au total)



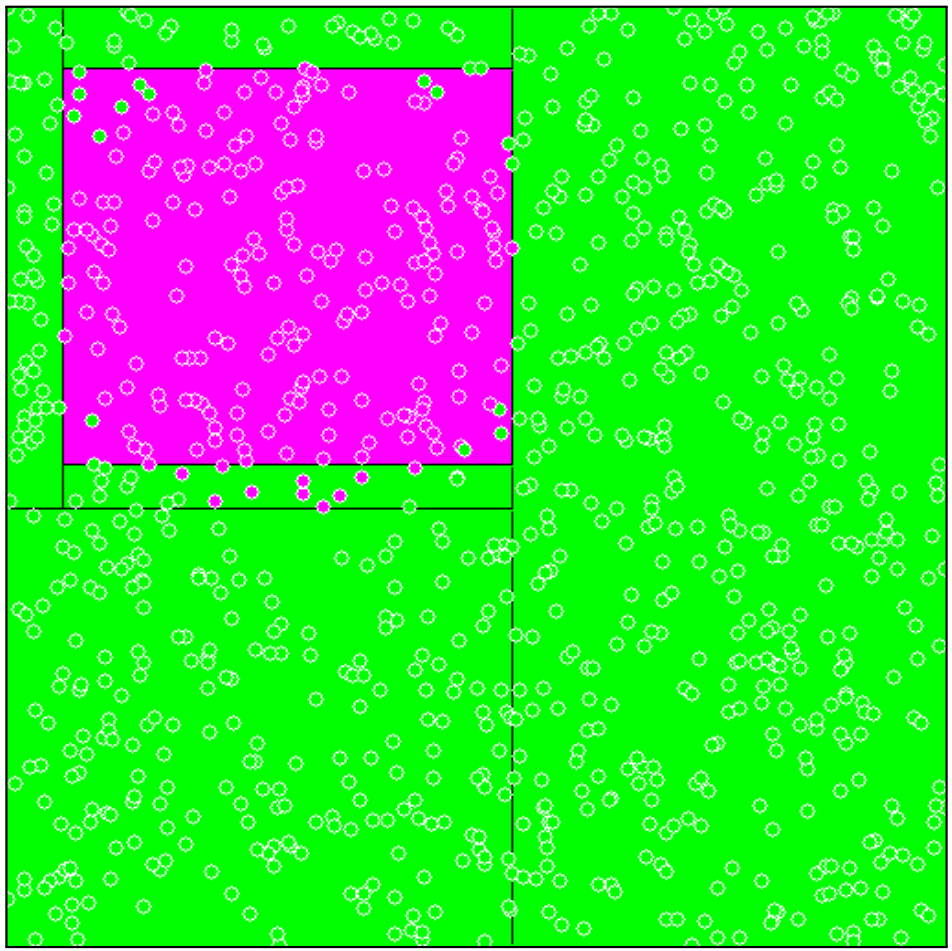
Profondeur : 3 (1 nouveau split, 4 feuilles au total)



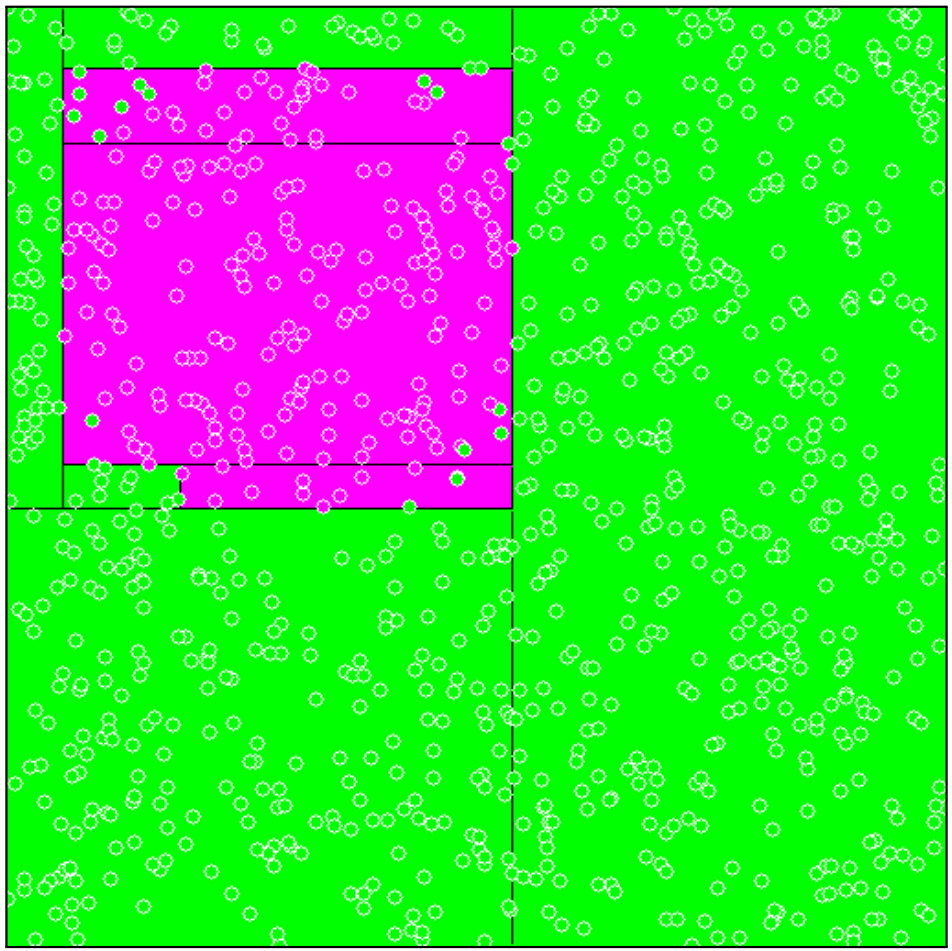
Profondeur : 4 (1 nouveau split, 5 feuilles au total)



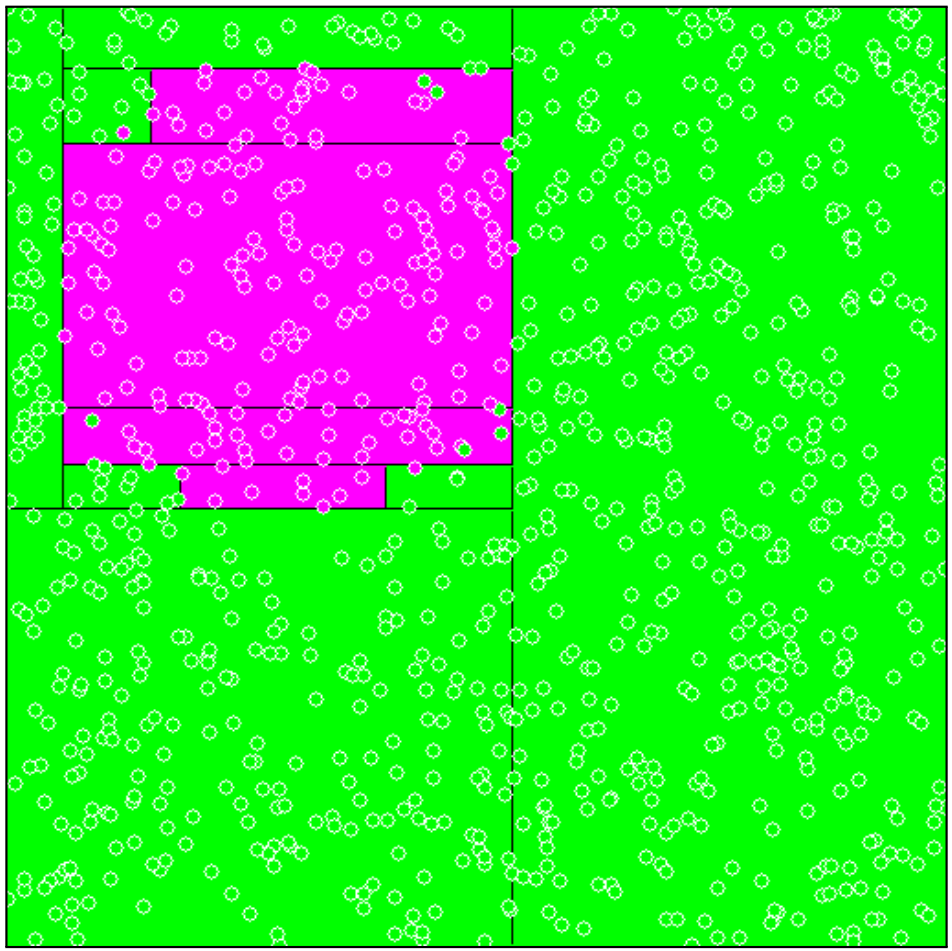
Profondeur : 5 (1 nouveau split, 6 feuilles au total)



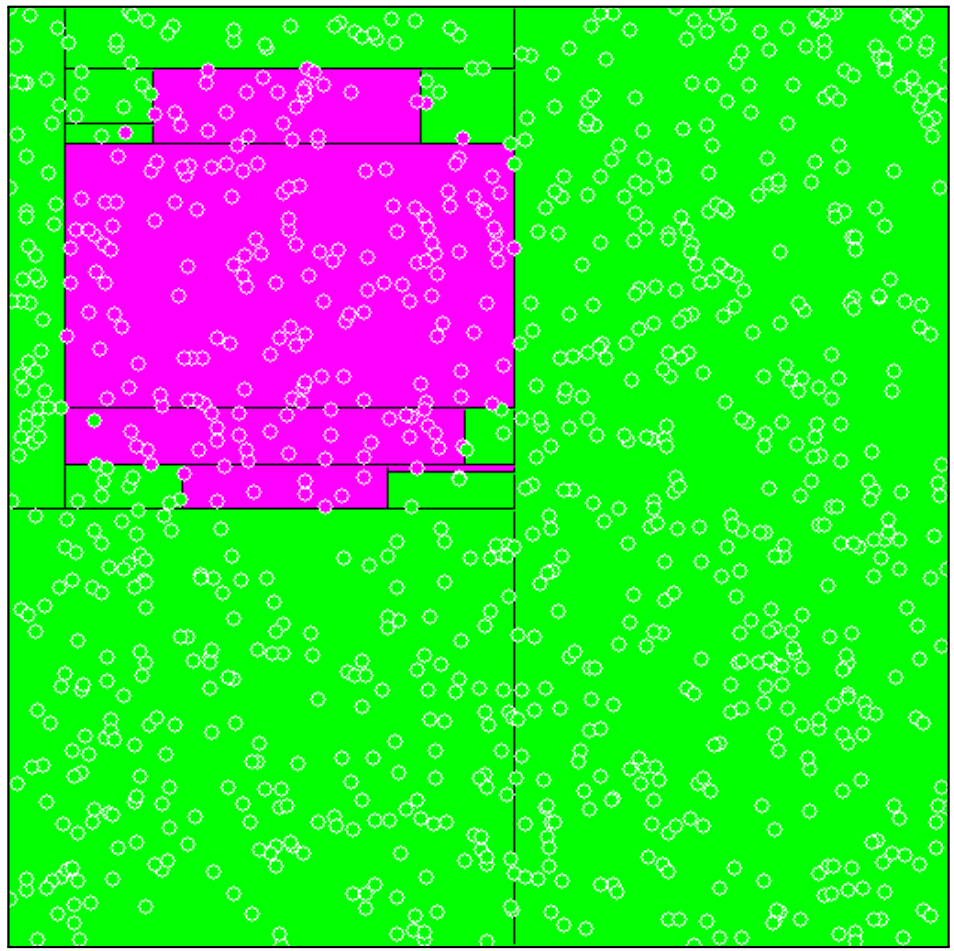
Profondeur : 6 (2 nouveaux splits, 8 feuilles au total)



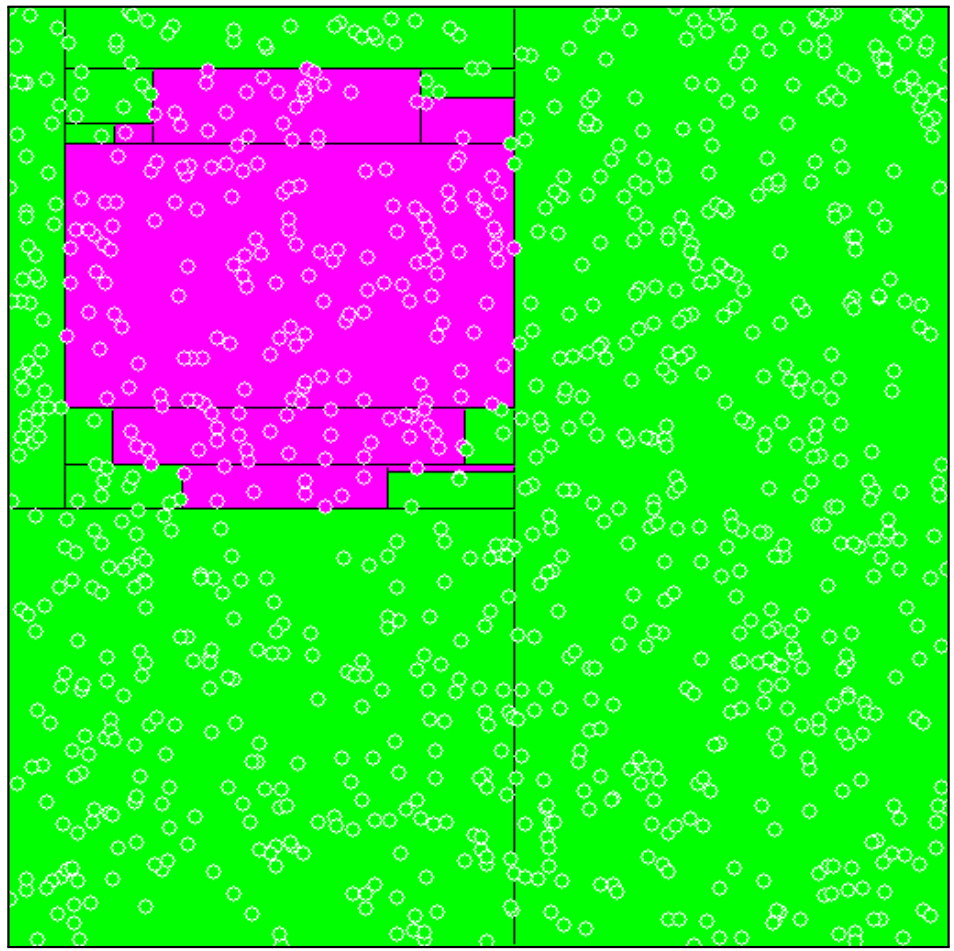
Profondeur : 7 (3 nouveaux splits, 11 feuilles au total)



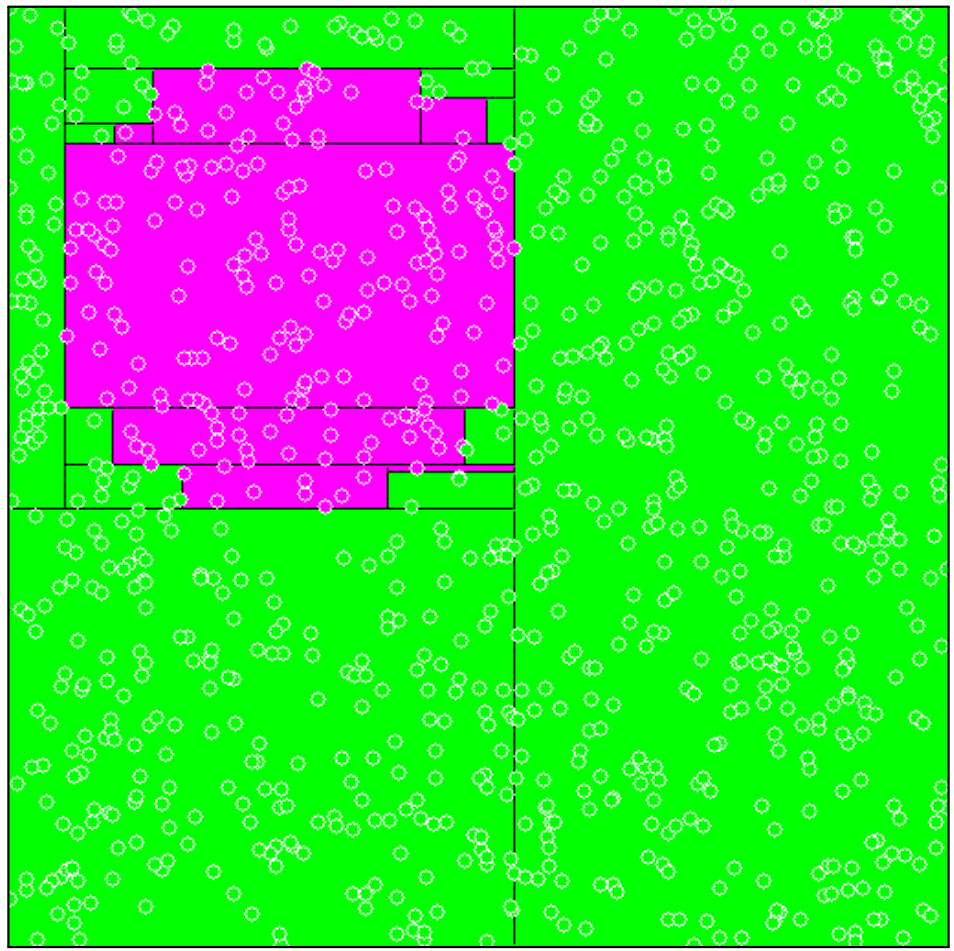
Profondeur : 8 (4 nouveaux splits, 15 feuilles au total)



Profondeur : 9 (3 nouveaux splits, 18 feuilles au total)



Profondeur : 10 (1 nouveau split, 19 feuilles au total)



Exemple de code d'arbre de décision dans scikit-learn (import des librairies nécessaires)

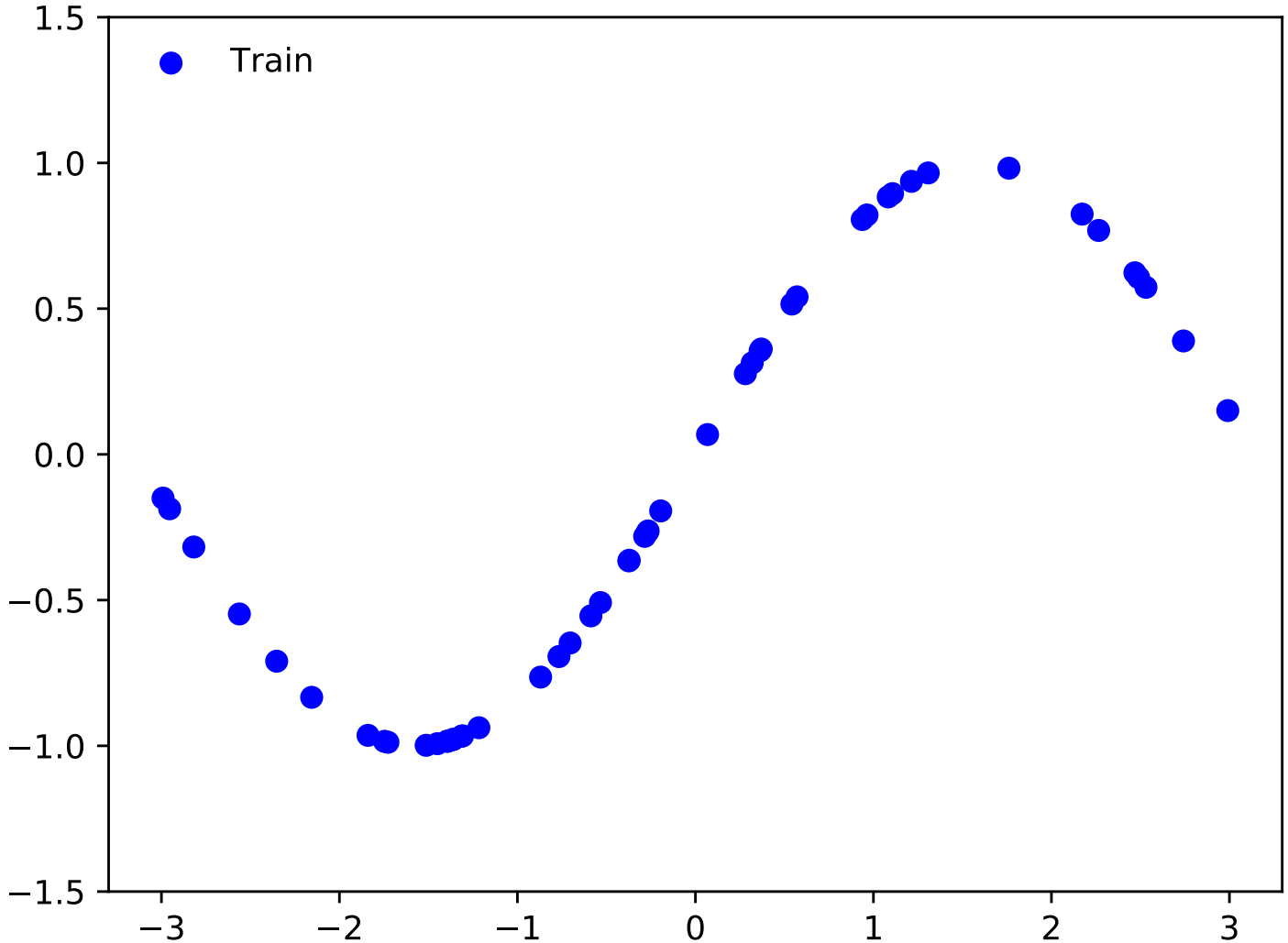
```
import math
import numpy as np
from sklearn import tree, ensemble
import matplotlib.pyplot as plt
```

Exemple de code d'arbre de décision dans scikit-learn (construction du jeu d'entraînement et visualisation)

```
x_train = np.random.uniform(-math.pi, math.pi, (50, 1))
y_train = np.sin(x_train)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.set_ylim(-1.5, 1.5)
ax.scatter(x_train, y_train, color = 'blue', label = 'Train')
```

Données d'entraînement



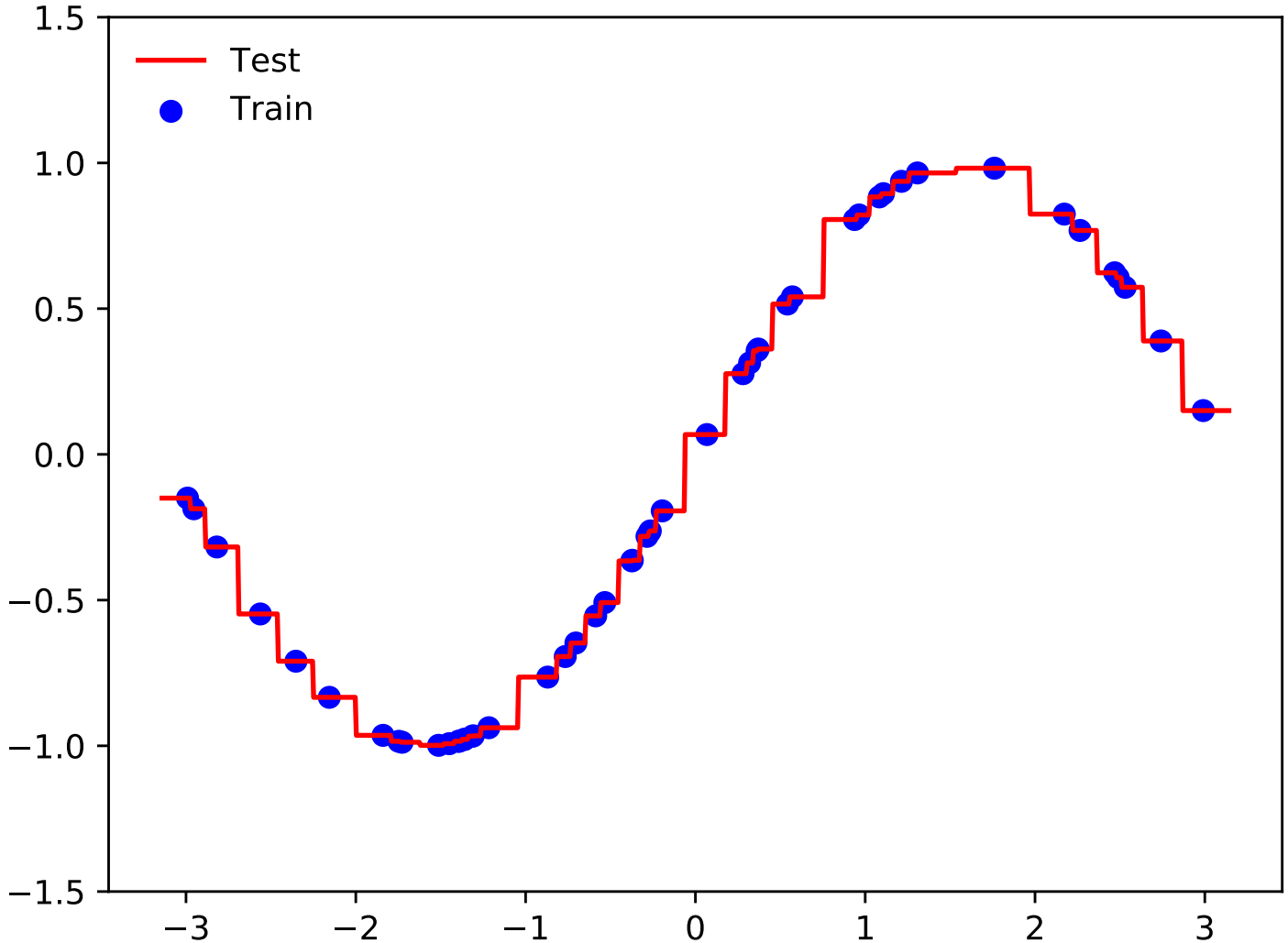
Exemple de code d'arbre de décision dans scikit-learn (entraînement et test)

```
dtr = tree.DecisionTreeRegressor()
model = dtr.fit(x_train, y_train)

x_test = np.linspace(-math.pi, math.pi, 1000).reshape(-1, 1)
y_test = model.predict(x_test)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.set_ylim(-1.5, 1.5)
ax.scatter(x_train, y_train, color = 'blue', label = 'Train')
ax.plot(x_test, y_test, color = 'red', label = 'Test')
```

Données d'entraînement et prédictions sur jeu de test



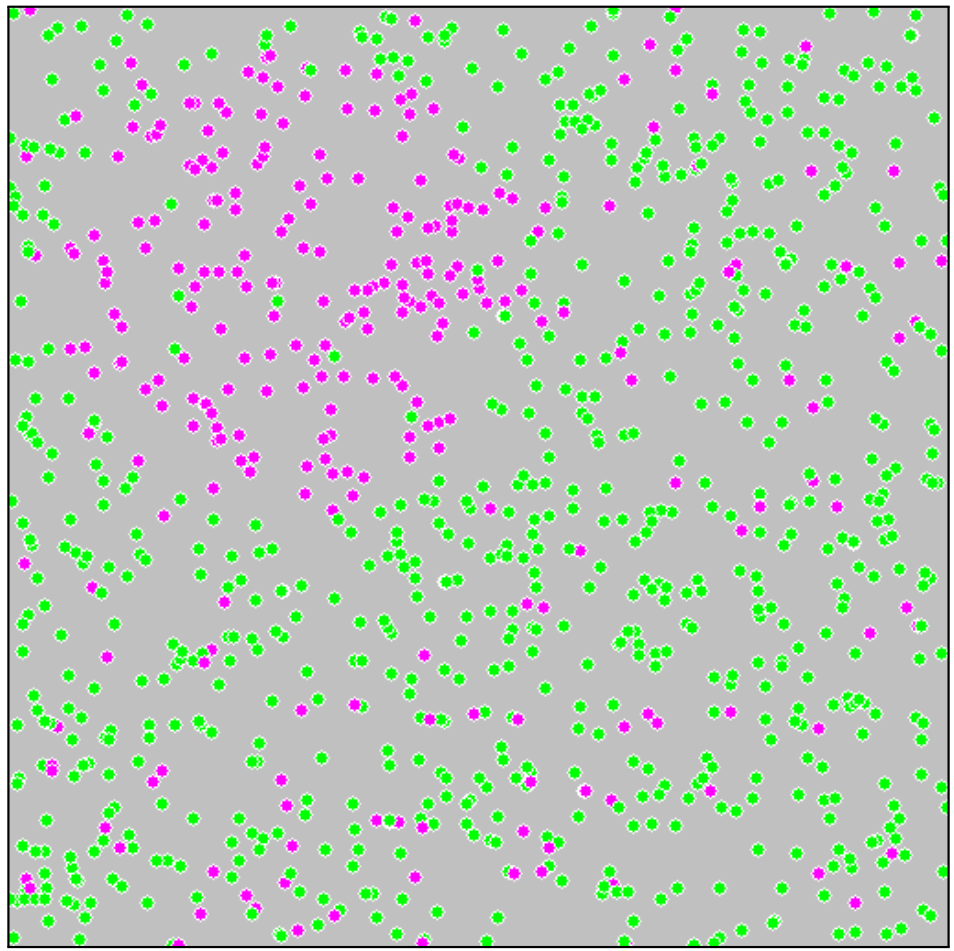
Arbres, régression et sur-apprentissage

La flexibilité des arbres de décision souffre du fait qu'ils ne régularisent pas "naturellement".

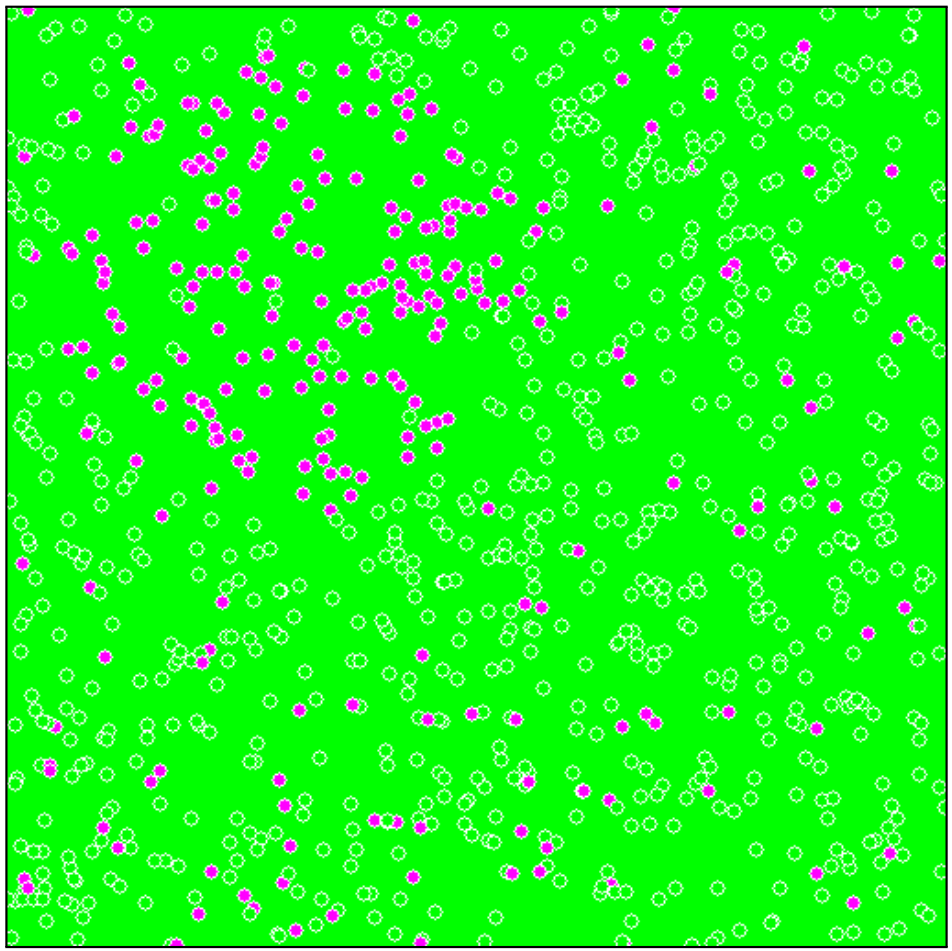
Si les labels sont par exemple bruités, la fonction obtenue peut-être particulièrement dégradée.

Exemple d'apprentissage CART sur données bruitées

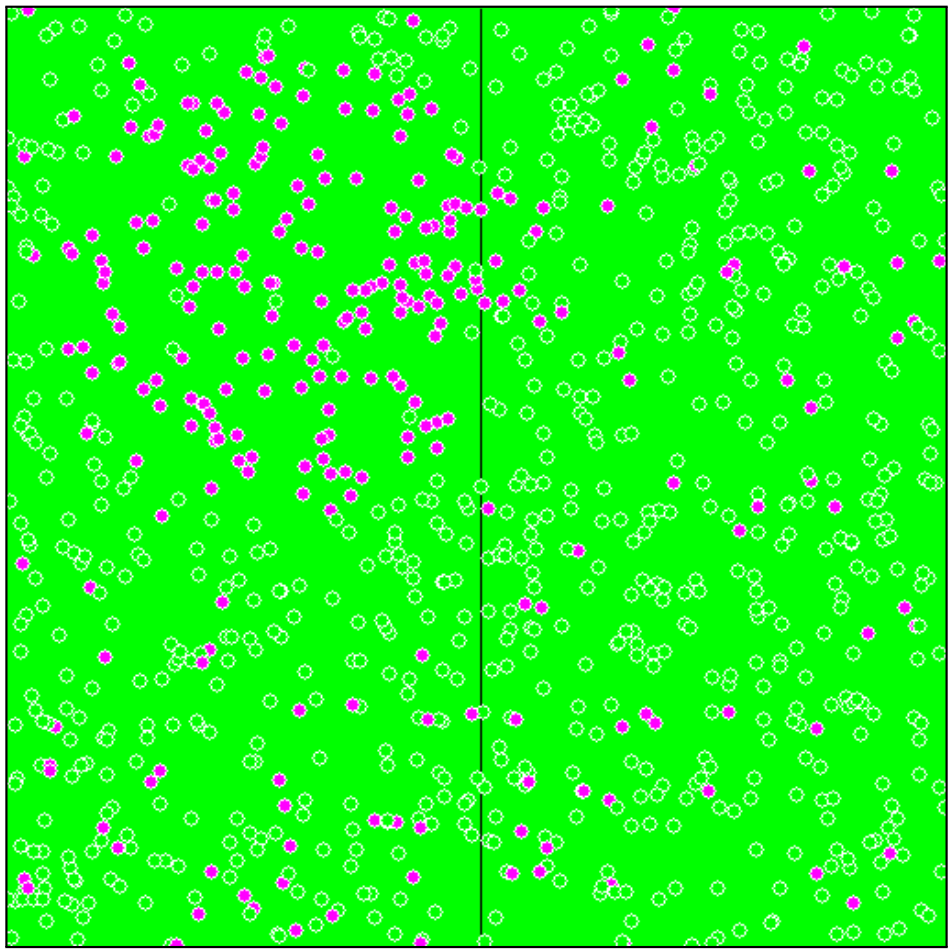
Nous pouvons reprendre notre exemple en ajoutant un "flip noise" dans les labels.



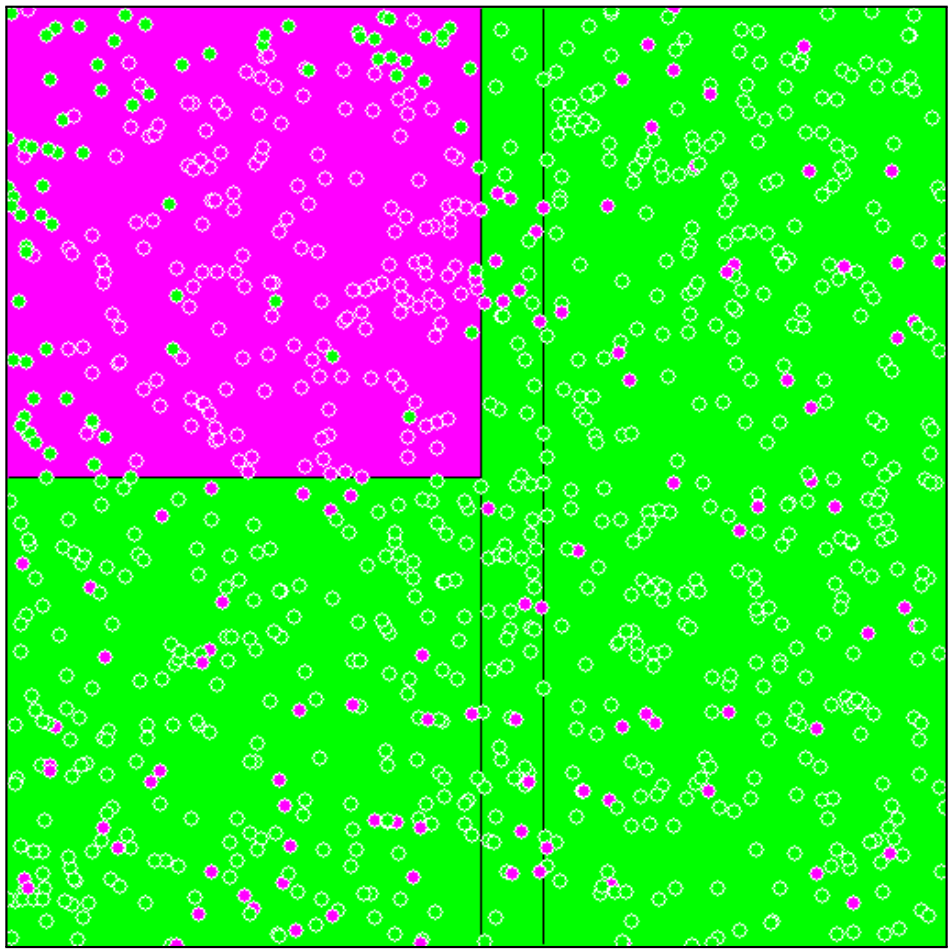
Profondeur : 0



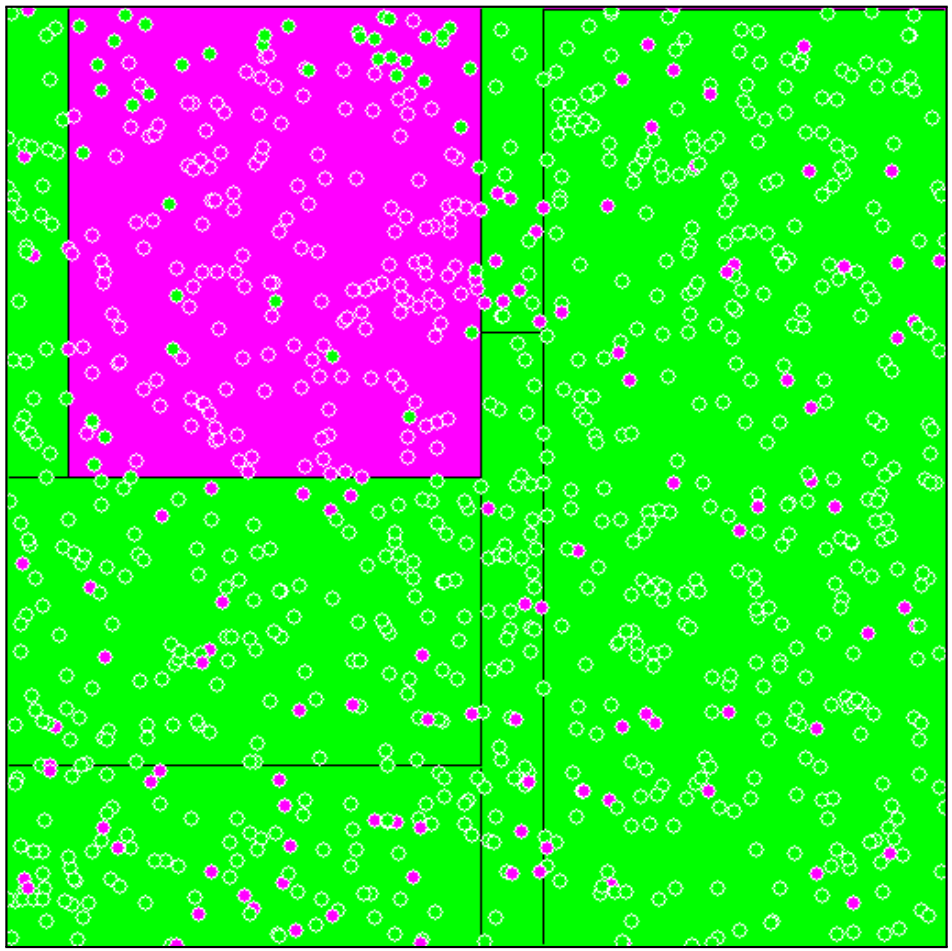
Profondeur : 1



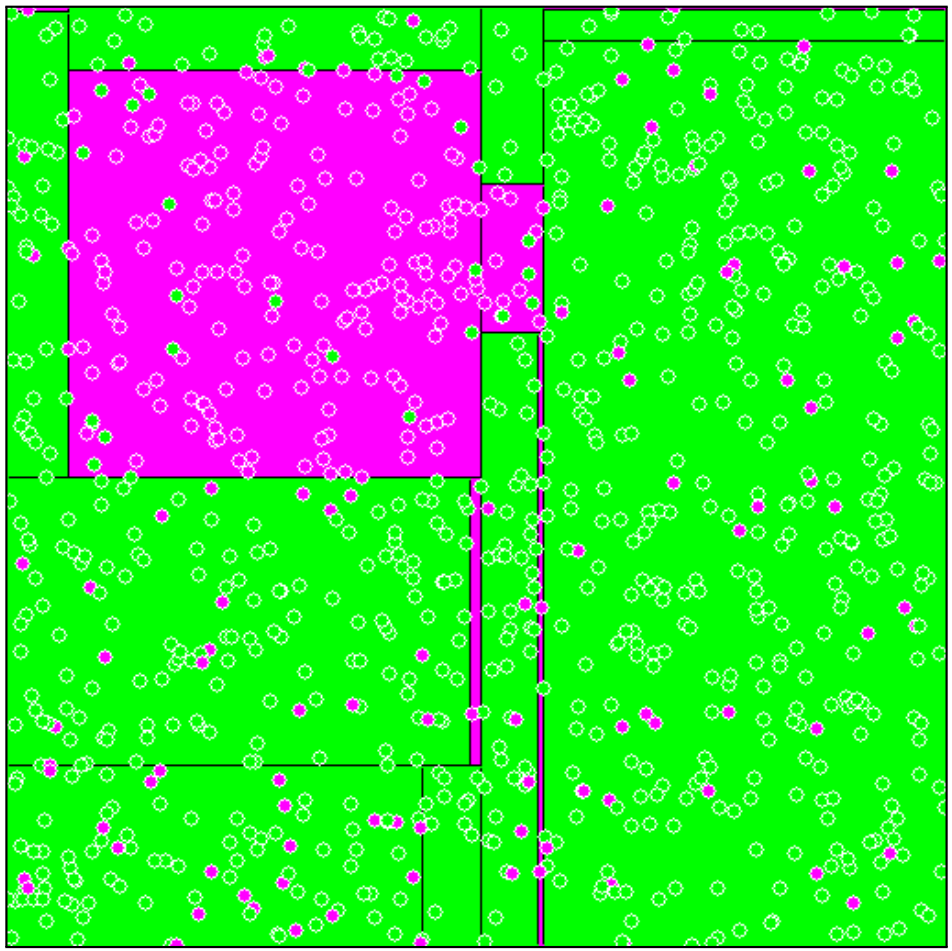
Profondeur : 2



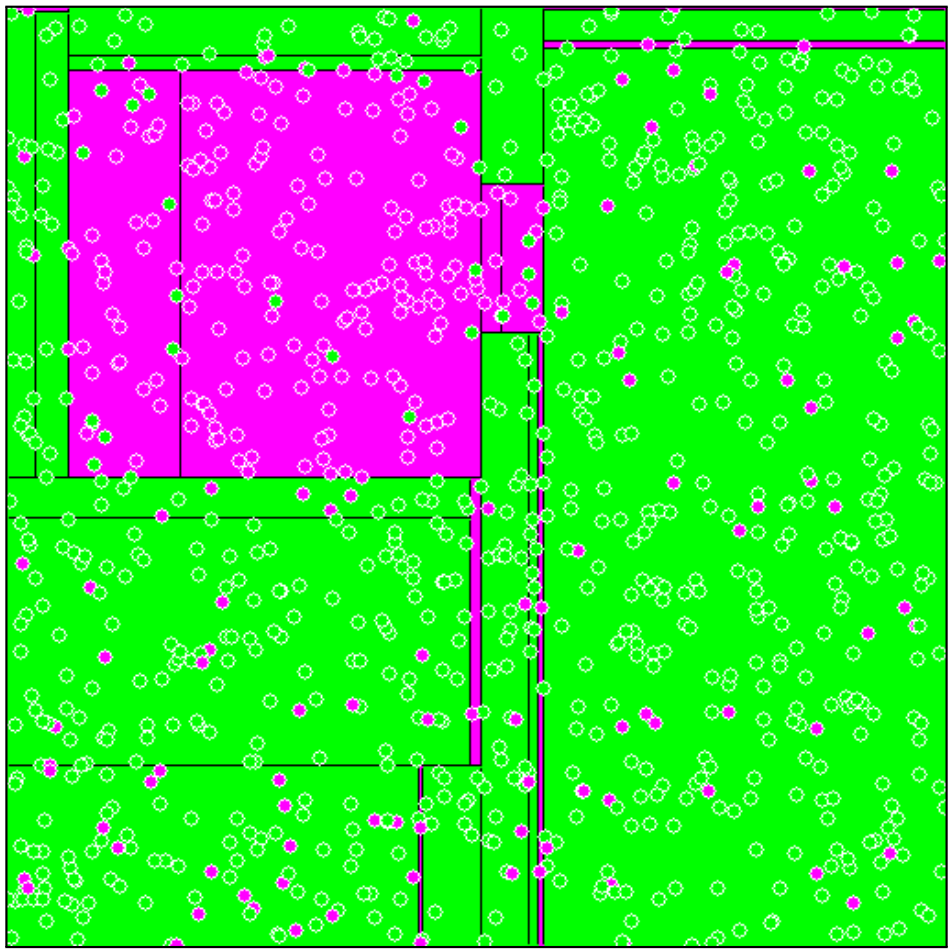
Profondeur : 3



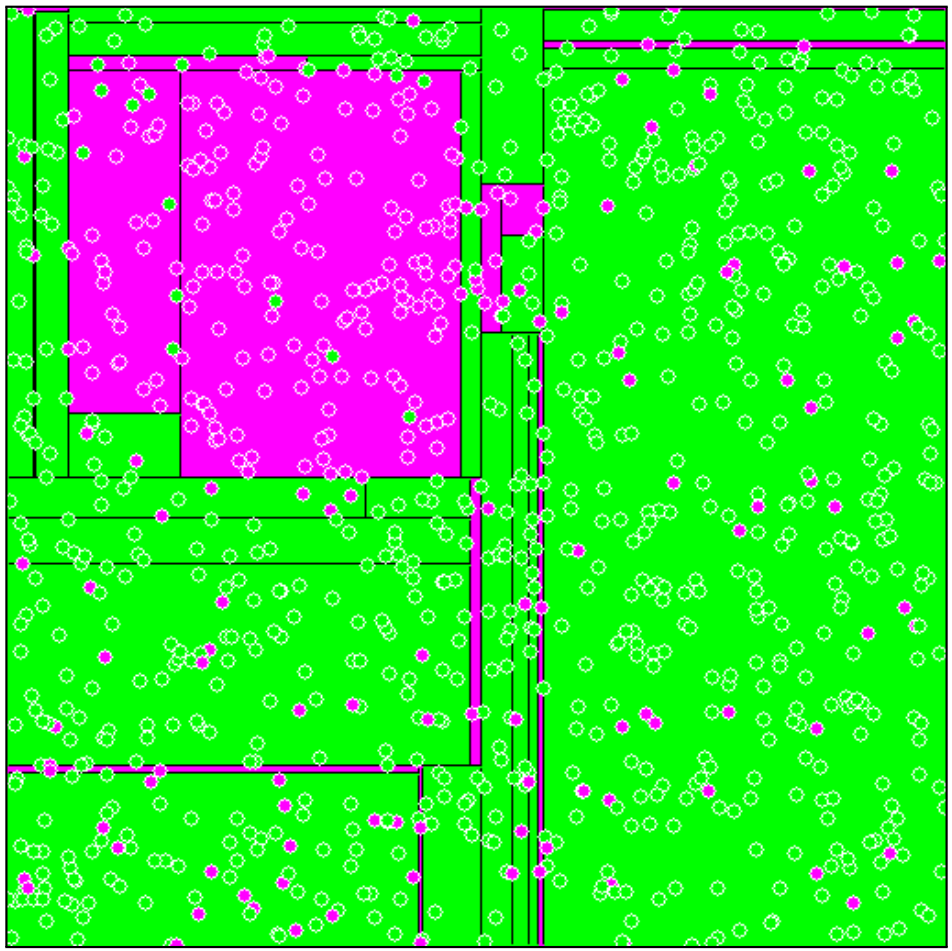
Profondeur : 4



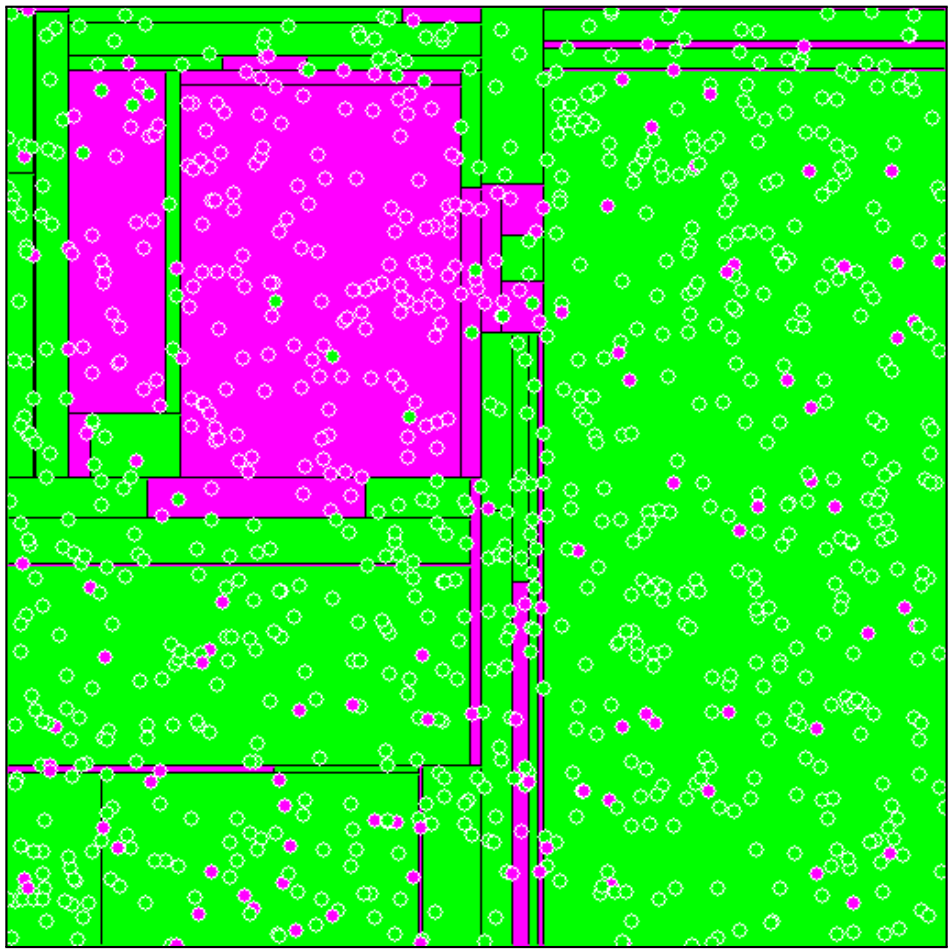
Profondeur : 5



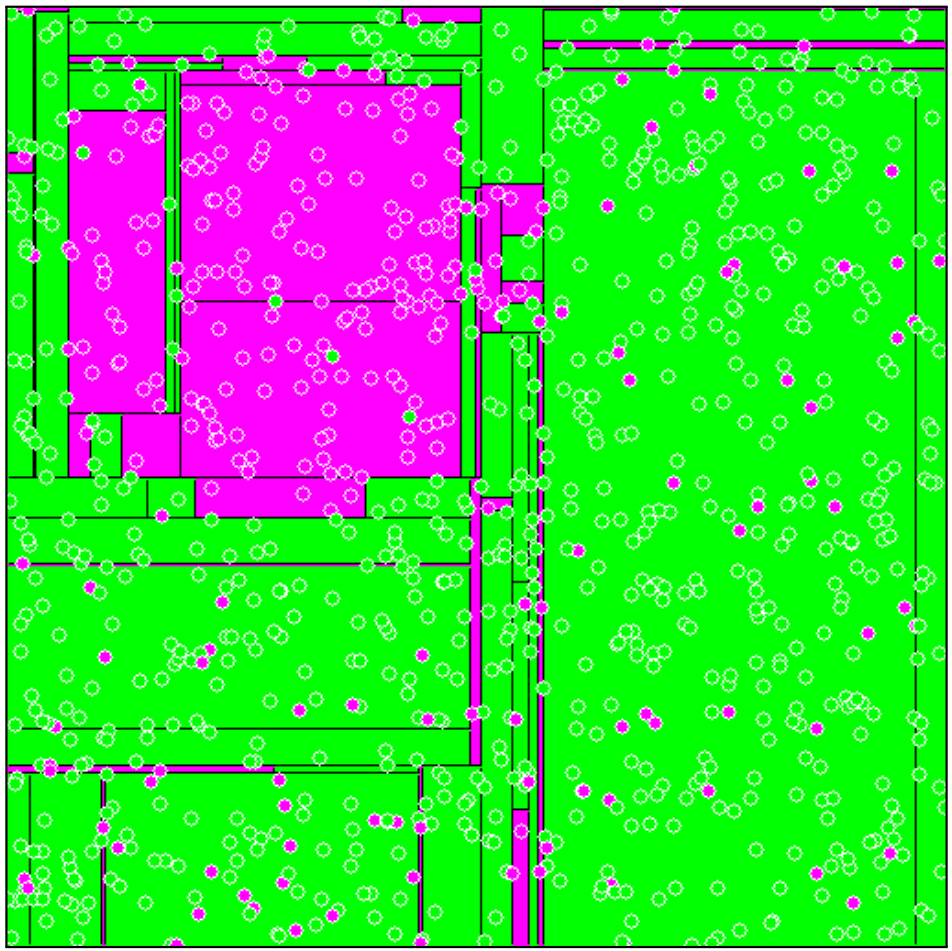
Profondeur : 6



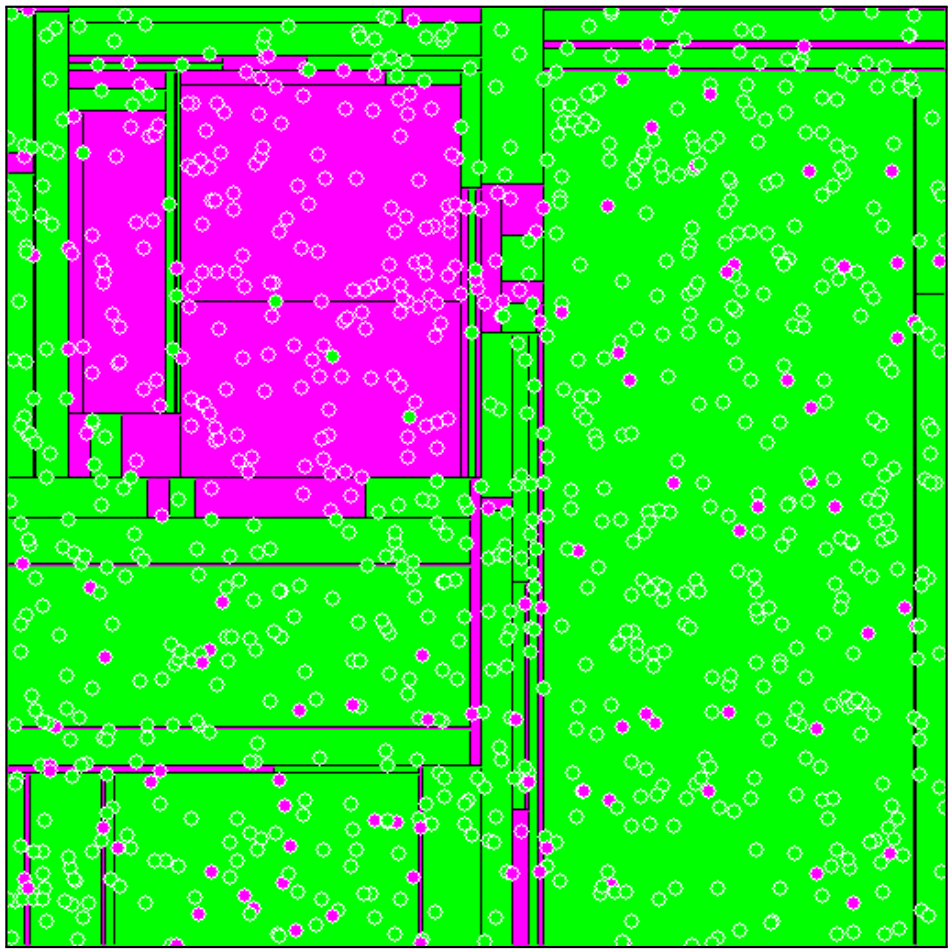
Profondeur : 7



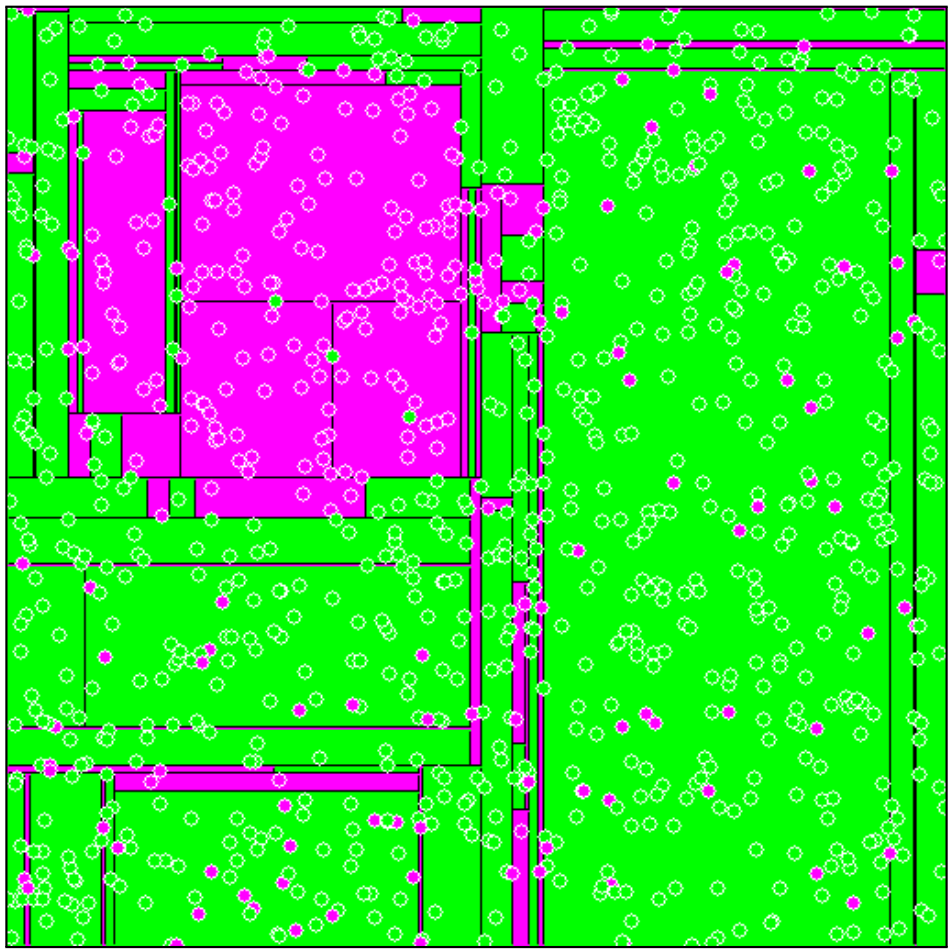
Profondeur : 8



Profondeur : 9



Profondeur : 10

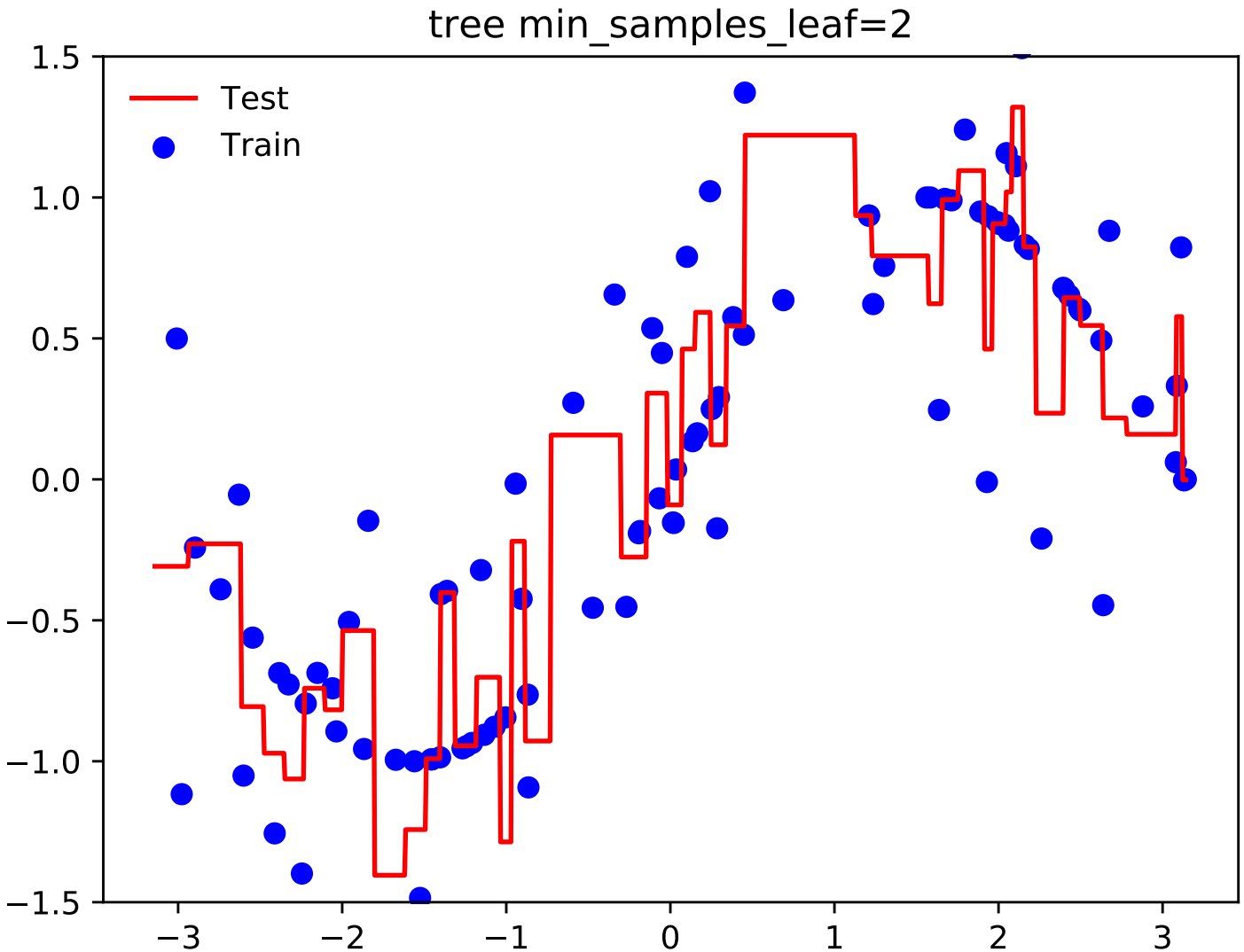


Régularisation en fixant nombre minimum d'exemples par feuille

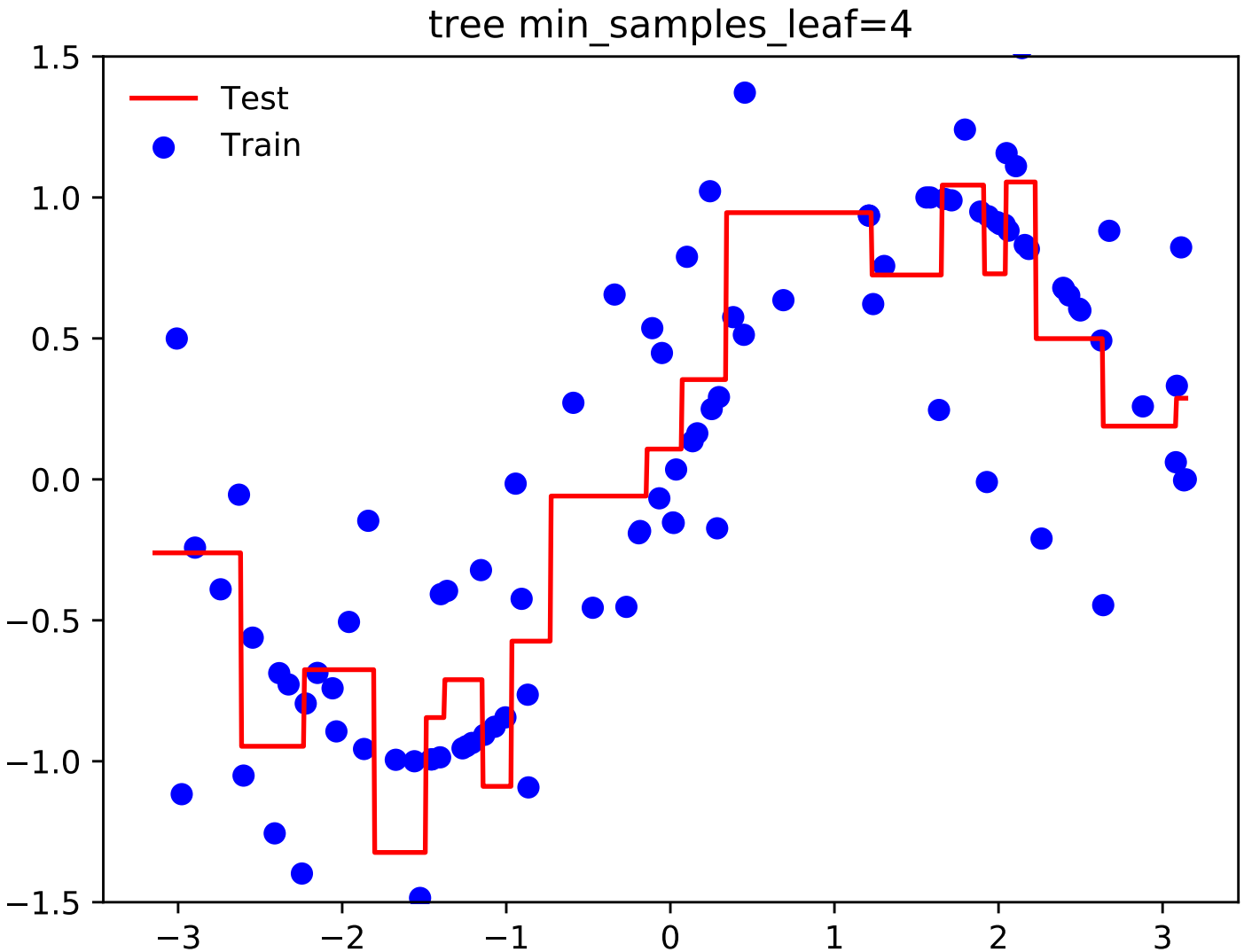
Le nombre minimum d'exemples nécessaires pour créer une feuille peut être augmenté pour que l'apprentissage soit plus stable en cas de bruit dans les données

```
sklearn.tree.DecisionTreeRegressor(min_samples_leaf = min_samples_leaf)
```

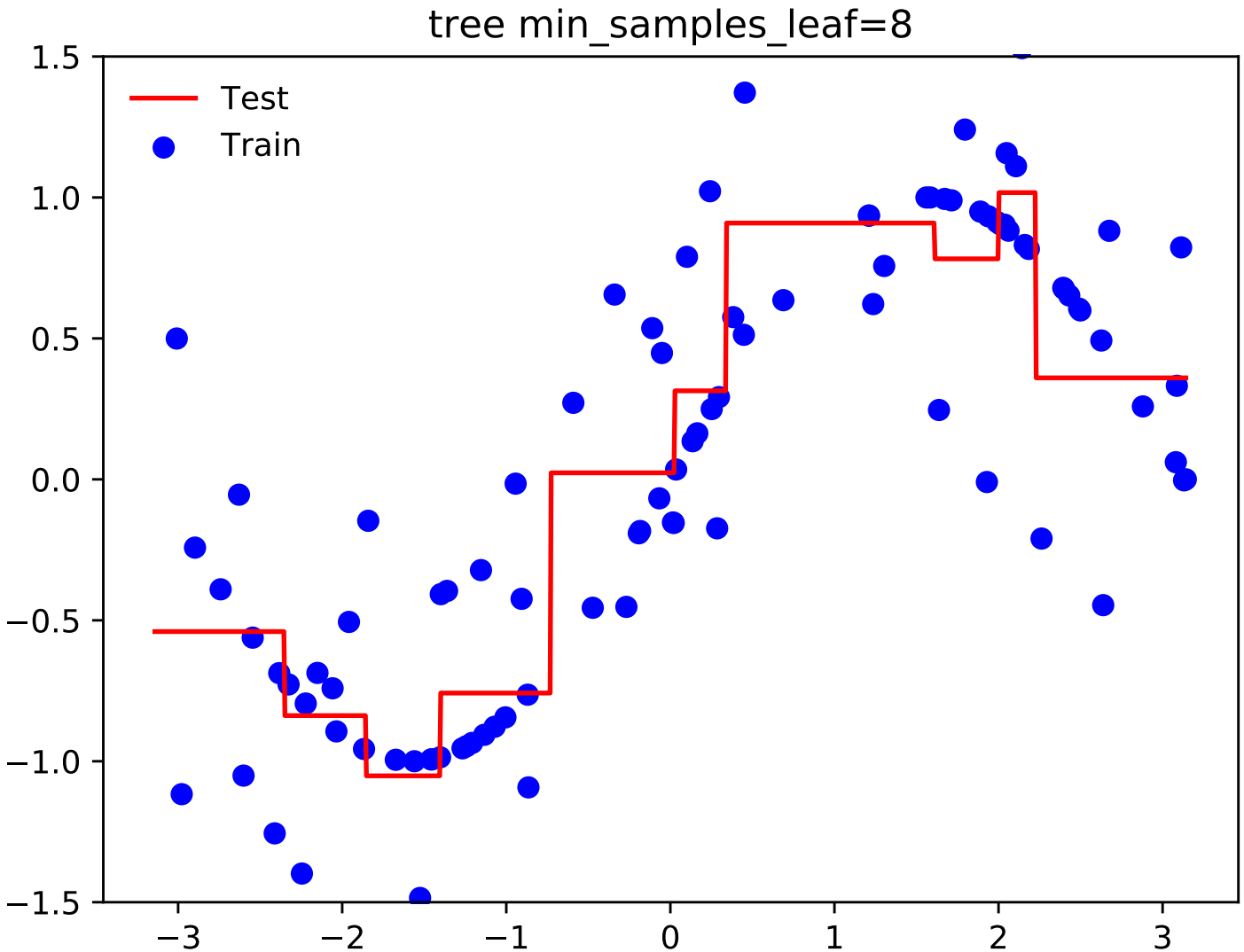
Apprentissage avec régularisation, bruit aléatoire sur 10% des données



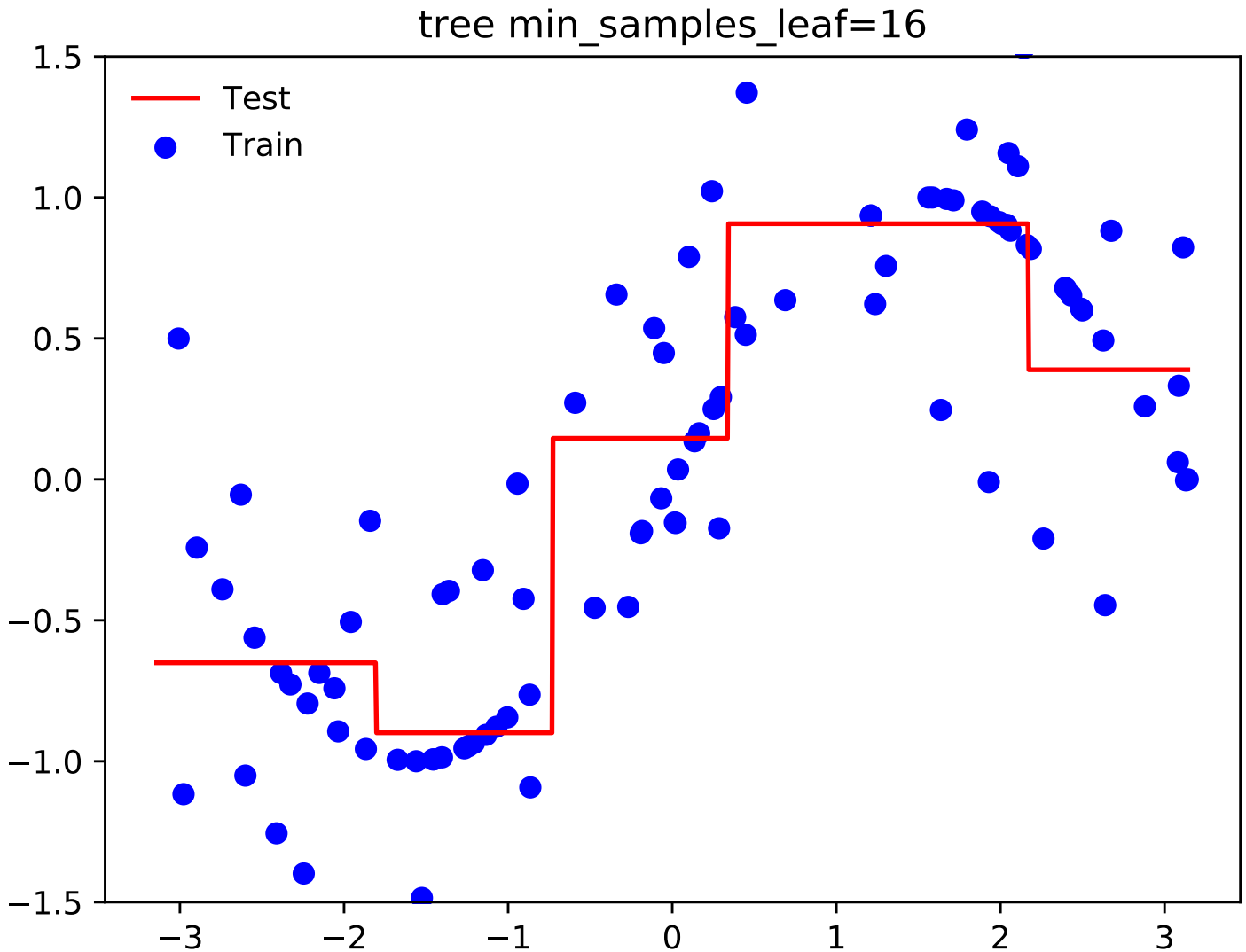
Apprentissage avec régularisation, bruit aléatoire sur 10% des données



Apprentissage avec régularisation, bruit aléatoire sur 10% des données



Apprentissage avec régularisation, bruit aléatoire sur 10% des données



Méthodes ensemblistes

https://www.srf.ch/news/landsgemeinde-lange-tradition-der-landsgemeinden

Méthodes ensemblistes

Problème : Comment gérer le sur-apprentissage de manière générale ?
Idée : faire des ensembles de prédicteurs (pour les arbres, ce seront des forêts)

Méthodes ensemblistes combinent de nombreux apprenants faibles dans le but d'obtenir une performance largement supérieure aux performances individuelles de ces apprenants faibles, car leurs erreurs se compensent les unes les autres.

On sépare 2 grandes catégories :

-  méthodes parallèles (bagging) : apprenants sont entraînés indépendemment
-  méthodes séquentielles (boosting) : apprenants sont entraînés itérativement

Méthodes ensemblistes : idées de base

Une manière plus efficace de réduire le sur-apprentissage consiste à construire plusieurs arbres, ou autres classificateurs, de manière à ce qu'*ils se trompent différemment*, et à combiner leurs prédictions.

On considère le vote majoritaire pour la classification, et la moyenne pour la régression.

On peut également estimer un indice de confiance, par exemple la proportion d'arbres faisant la même prédiction pour la classification, et la variance empirique des prédictions pour la régression.

Forêts obtenues par bagging

La méthode la plus classique, dite de *bagging*, entraîne plusieurs arbres séparément, c'est à dire sans tenir compte de leur comportement joint. Ces arbres forment alors une forêt.

Il est crucial de rajouter une composante aléatoire dans le processus d'apprentissage de manière à produire des arbres différents qui se trompent différemment.

Nous regardons 2 manières d'incorporer l'aspect aléatoire :

- forêt par bootstrapping : on génére plusieurs ensembles d'apprentissage par ré-échantillonage du jeu de données initial, sur lesquel on entraîne des arbres
- forêt aléatoire : on construit plusieurs arbres, à partir du même jeu de données, mais en injectant du hasard dans la construction de chaque arbre

Construction d'une forêt par bootstrapping (1/2)

La première technique proposée historiquement repose sur du "bootstrapping".

On simule un ensemble d'apprentissage différent pour chaque arbre en re-échantillonant l'ensemble complet uniformément.

Les ensembles ainsi produits ne sont donc évidemment pas indépendants, et les estimateurs statistiques que l'on utilisera devront être manipulés avec prudence.

Note :
"Pull oneself up by one's own bootstraps"
≈
"se tirer vers le haut par ses propres moyens"

accessworkwear.com.au

Construction d'une forêt par bootstrapping (2/2)

Plus formellement, on avait un ensemble

$$\mathscr{D} = \{(x^1, y^1), \ldots, (x^N, y^N)\}$$

à partir duquel on entraînait un simple arbre $\mathscr{T}$.

Pour entrainer T arbres $\mathscr{T}_t, t = 1, \ldots, T$ par **bootstrapping** :
1. on génère les $T \times N$ nombres

$$\tilde{n}_n^t \text{ i.i.d. uniformément sur } \{1, \ldots, N\},$$

avec $t = 1, \ldots, T$ et $n = 1, \ldots, N$ pour obtenir les T jeux

$$\mathscr{D}_t = \{(x^{\tilde{n}_1^t}, y^{\tilde{n}_1^t}), \ldots, (x^{\tilde{n}_N^t}, y^{\tilde{n}_N^t})\}, \quad t = 1, \ldots T$$

(*Note :* chaque jeu est obtenu par tirage de N observations *avec remise*)
2. on détermine T arbres $\mathscr{T}_t, t = 1, \ldots T$, chacun entraîné sur son jeu $\mathscr{D}_t$

Ces T arbres $\mathscr{T}_t, t = 1, \ldots T$ constituent une **forêt** par bootstrapping.

Prédiction à partir d'une forêt

Et la prédiction de cette *forêt* sera donc, pour la régression

$$\mathscr{F}(x) = \frac{1}{T} \sum_{t=1}^{T} \mathscr{T}_t(x)$$

et pour la classification on prend la classe dominante :

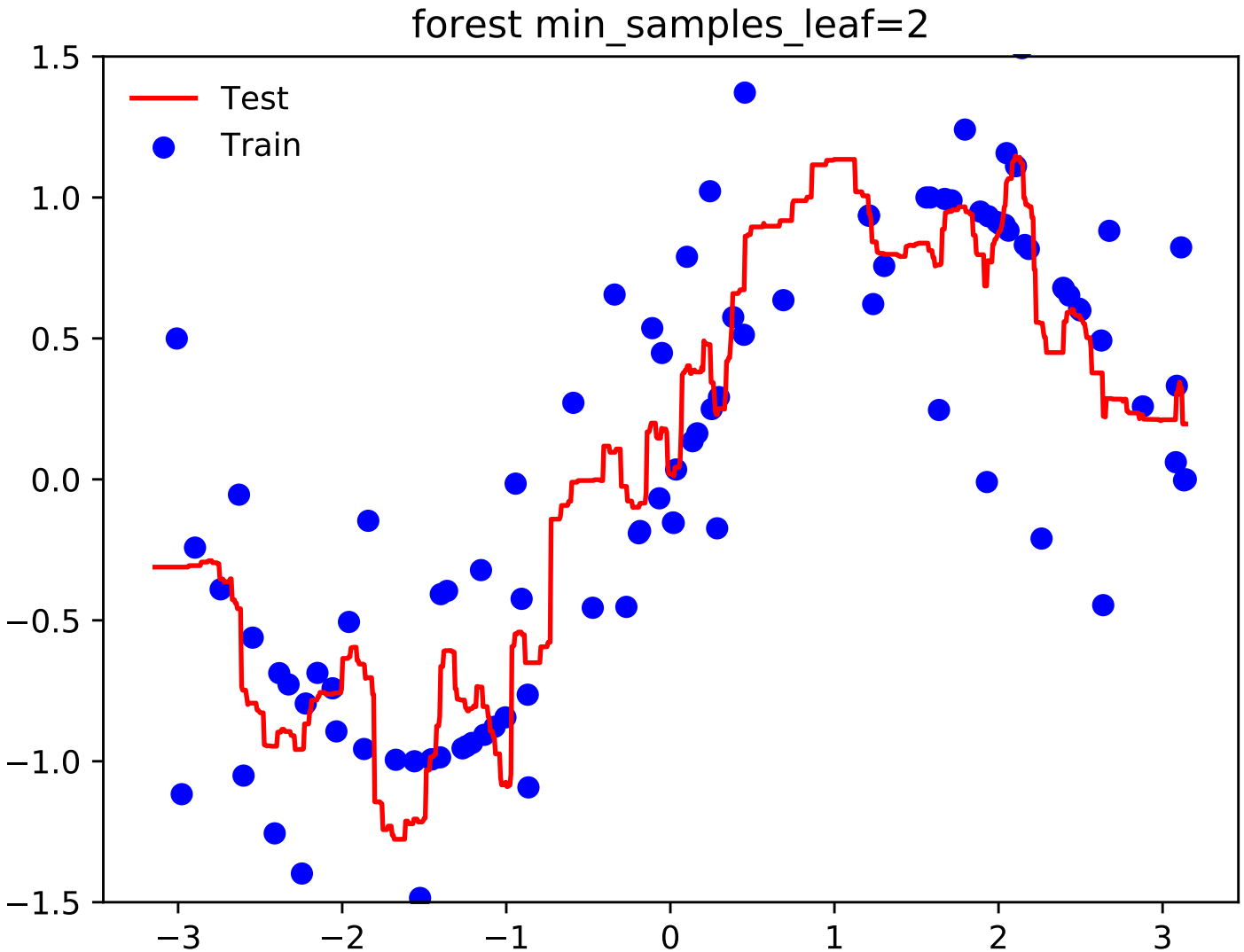
$$\mathscr{F}(x) = \underset{c}{\text{argmax}} \left| \{ t : \mathscr{T}_t(x) = c \} \right| .$$

Implémentation d'une forêt par bootstrapping avec scikit-learn

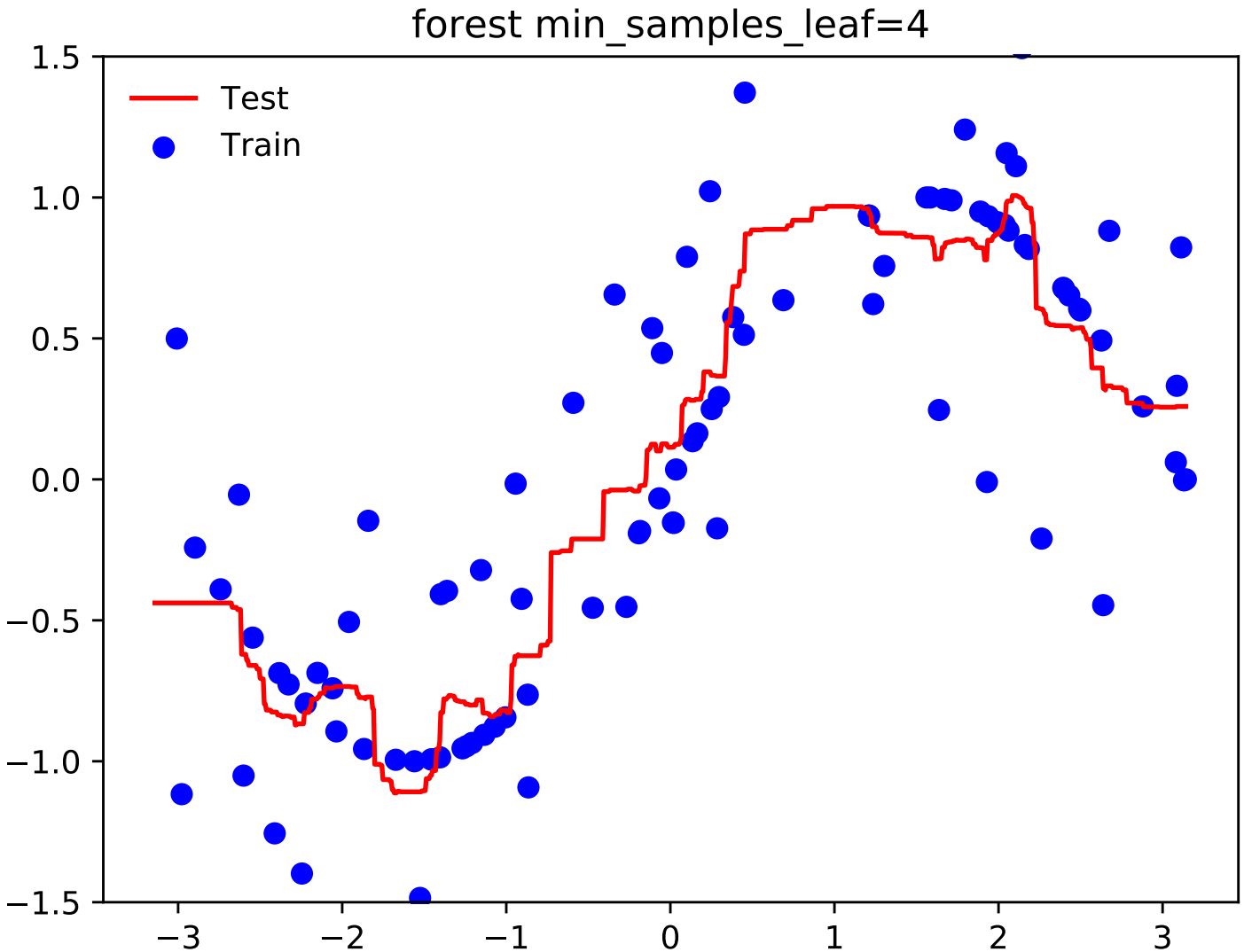
Par défaut, scikit-learn construit 100 arbres dans une forêt.

```
sklearn.ensemble.RandomForestRegressor(min_samples_leaf = min_samples_leaf)
```

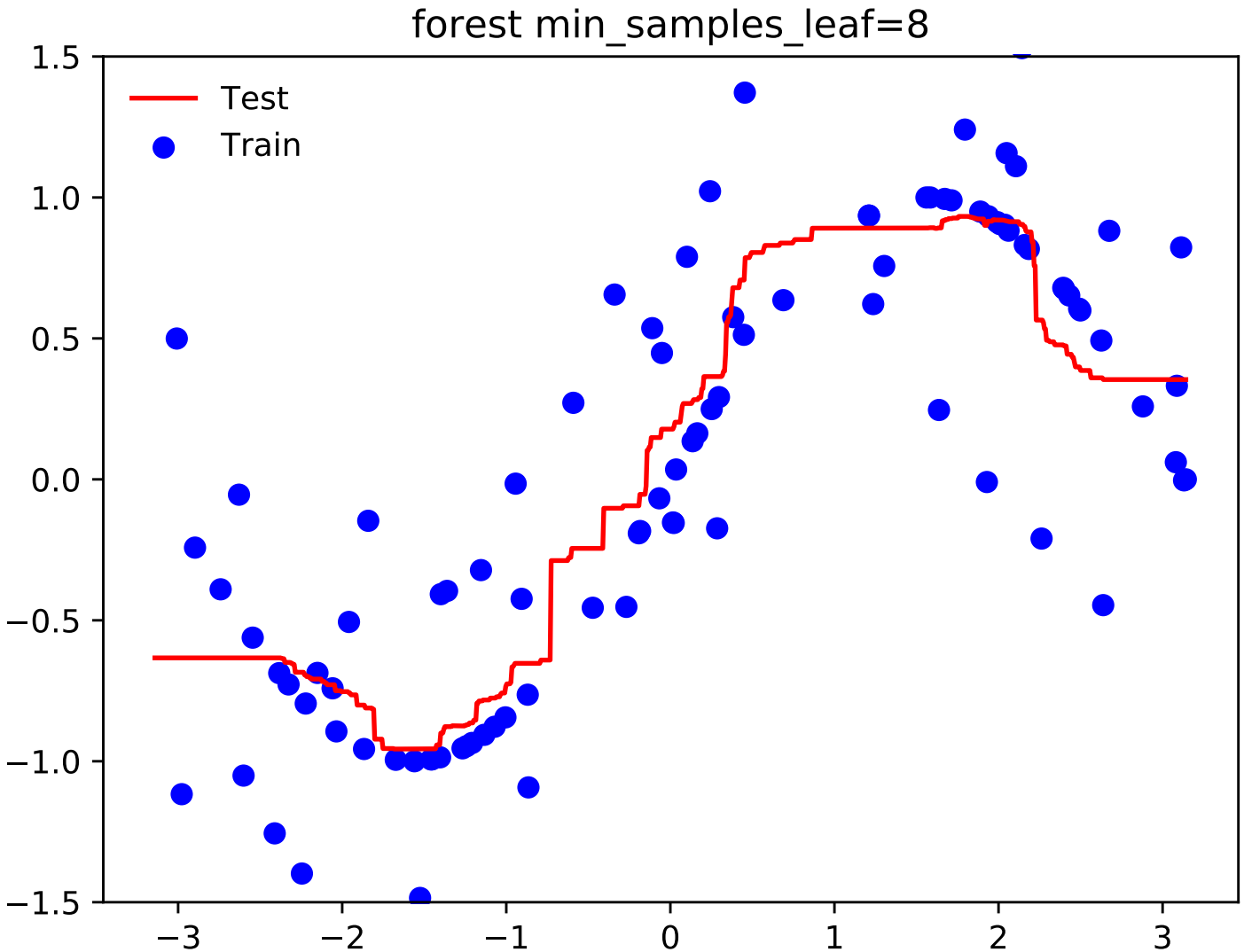
Exemple de prédiction par une forêt obtenue via bootstrapping



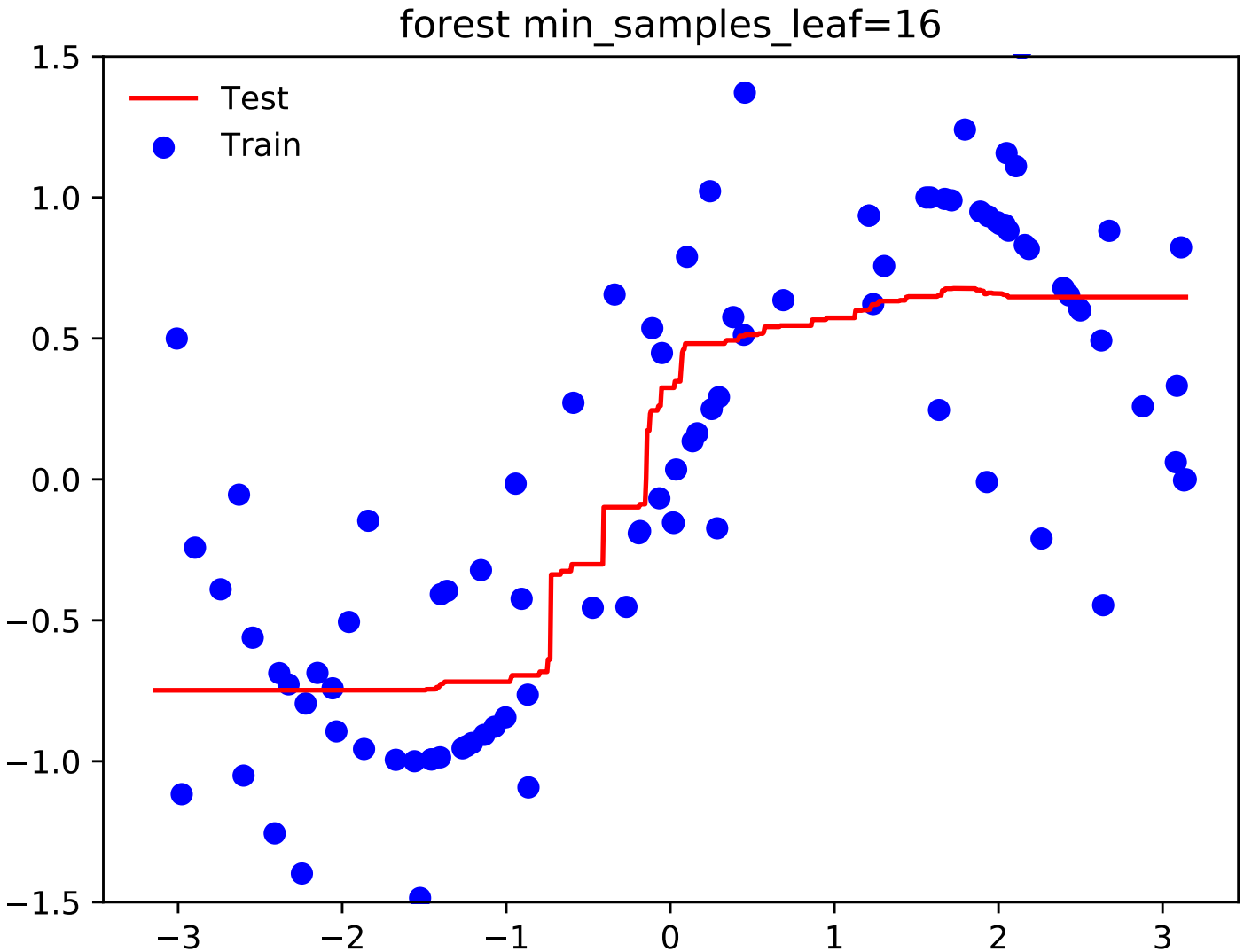
Exemple de prédiction par une forêt obtenue via bootstrapping



Exemple de prédiction par une forêt obtenue via bootstrapping



Exemple de prédiction par une forêt obtenue via bootstrapping



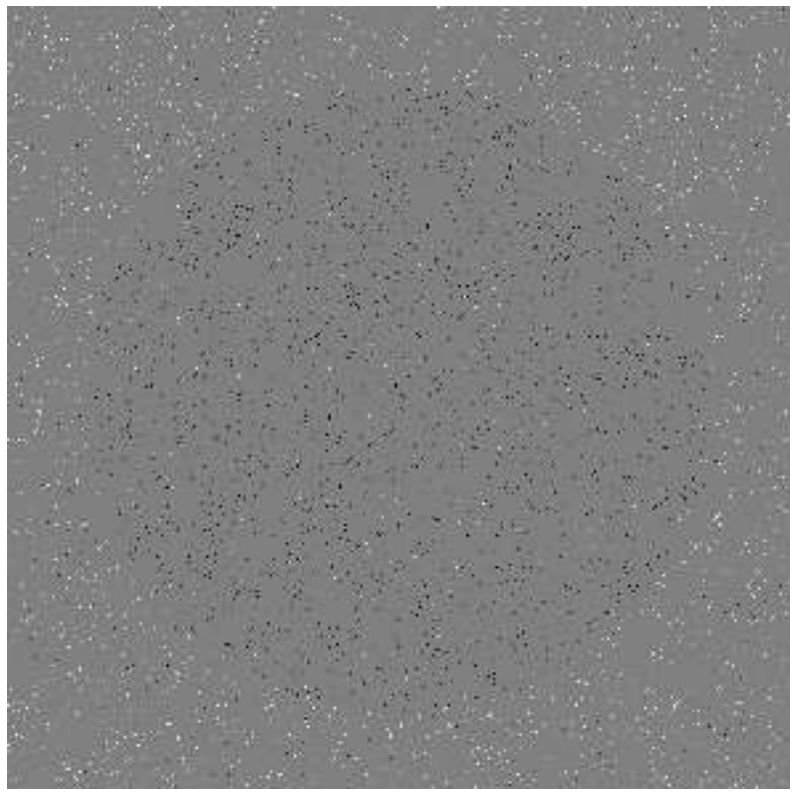
Forêt aléatoire (random forest)

Une alternative au bootstrapping consiste à utiliser pour chaque arbre toutes les données d'apprentissage, mais à injecter du hasard dans la construction de l'arbre.

La manière classique consiste à sélectionner aléatoirement un sous-ensemble de variables à chaque nœud qu'on considère durant la construction de l'arbre. On parle alors de "forêt aléatoire".

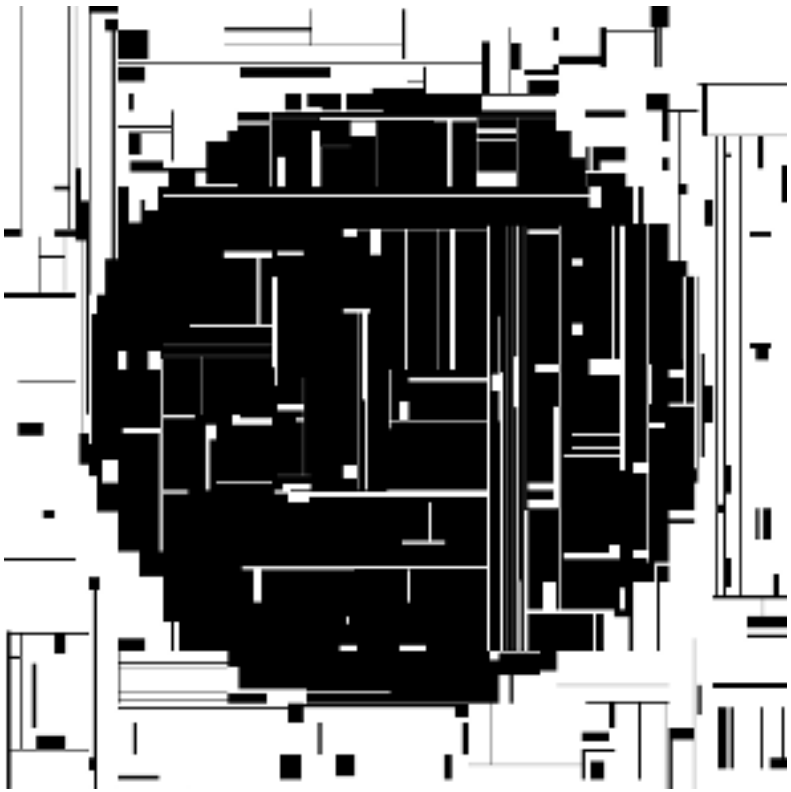
Example d'apprentissage avec forêt aléatoire

Ensemble d'apprentissage (bruité : 5% des label sont inversés)
(classe 1 (points noirs) : intérieur du disque,
classe 0 (points blancs) : extérieur du disque)



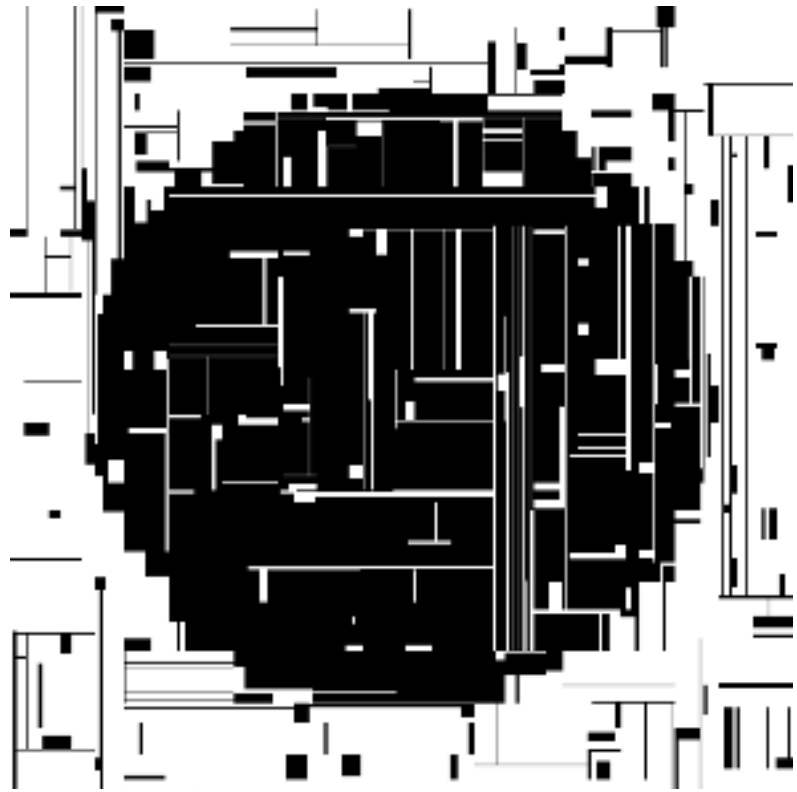
Un arbre

`min_samples_leaf = 1`



Probabilité

0 ou 1 : 1 exemple/feuille

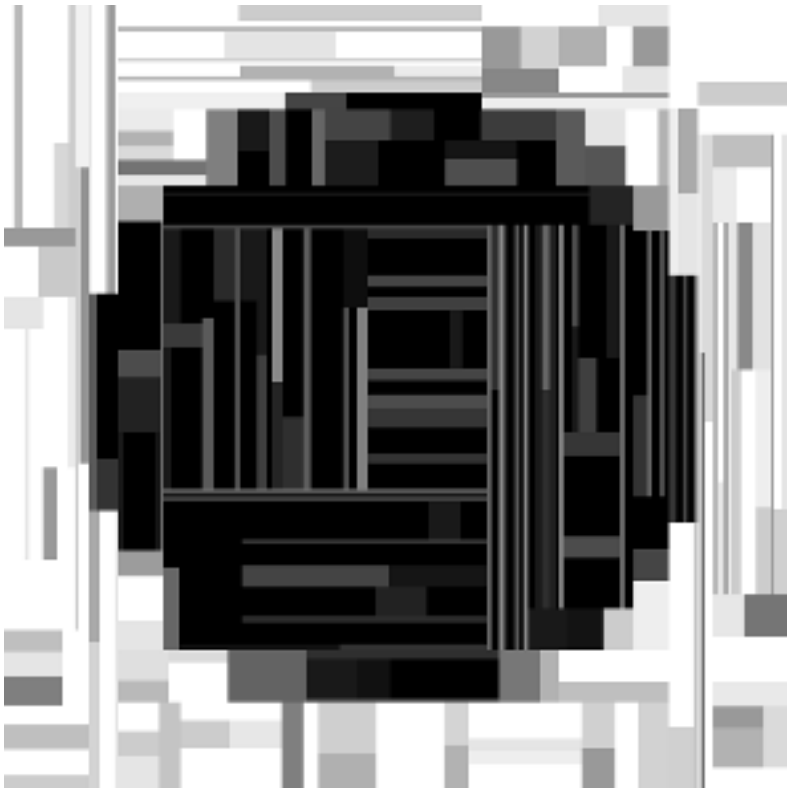


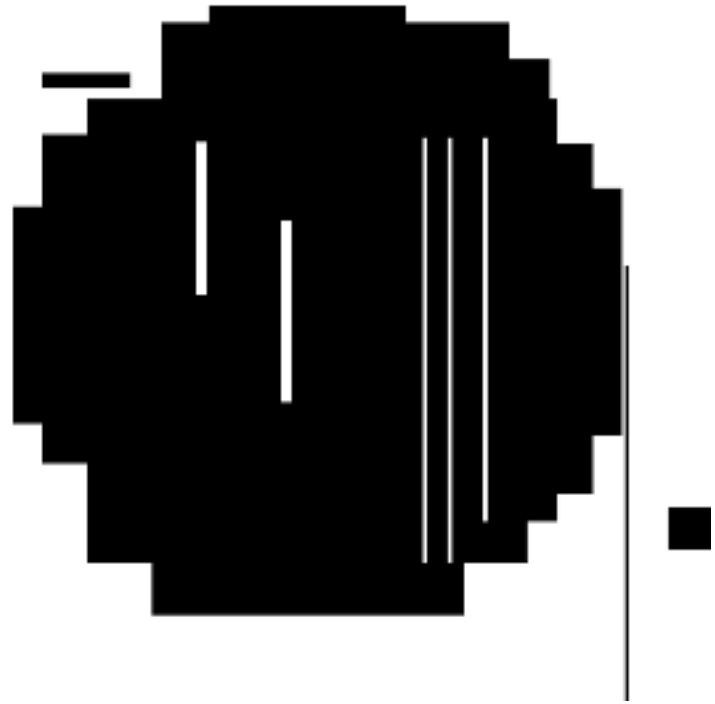
Prédiction

décision dure : classe 0 ou 1

Un arbre

`min_samples_leaf = 10`



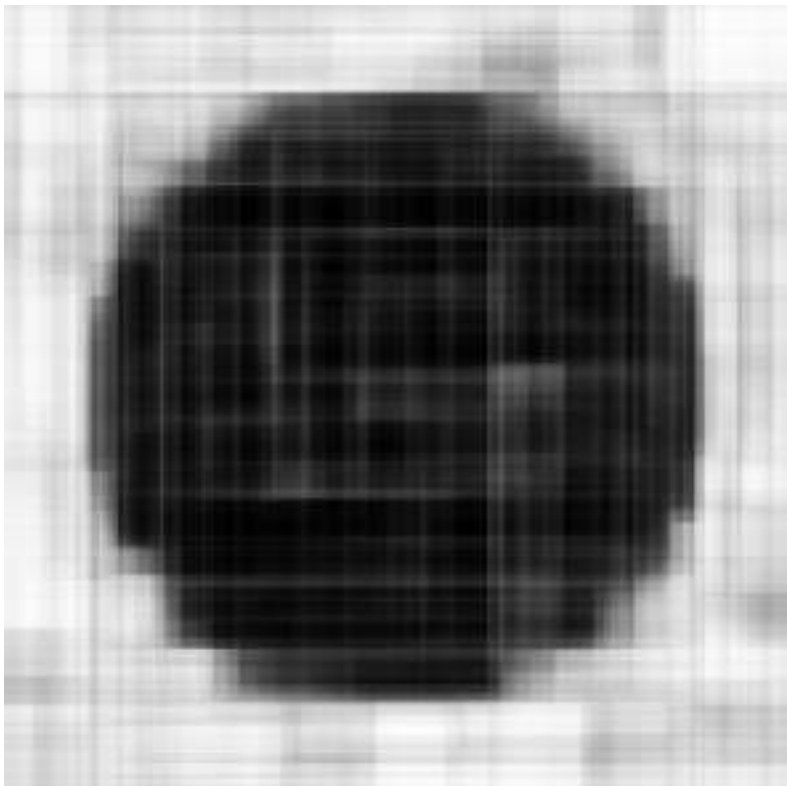


Probabilité entre 0 et 1
par pas de 0.1 : 10 exemples/feuille

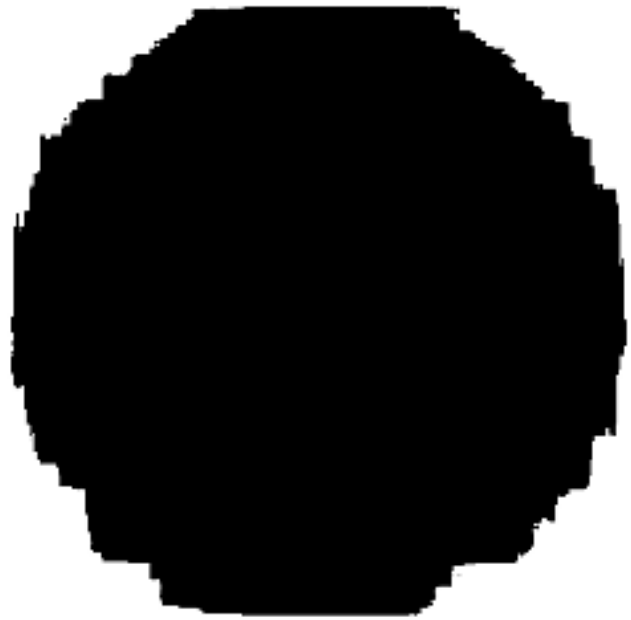
Prédiction
décision dure : classe 0 ou 1

Une forêt (obtenue par bootstrapping)

`min_samples_leaf = 10, bootstrap = True`



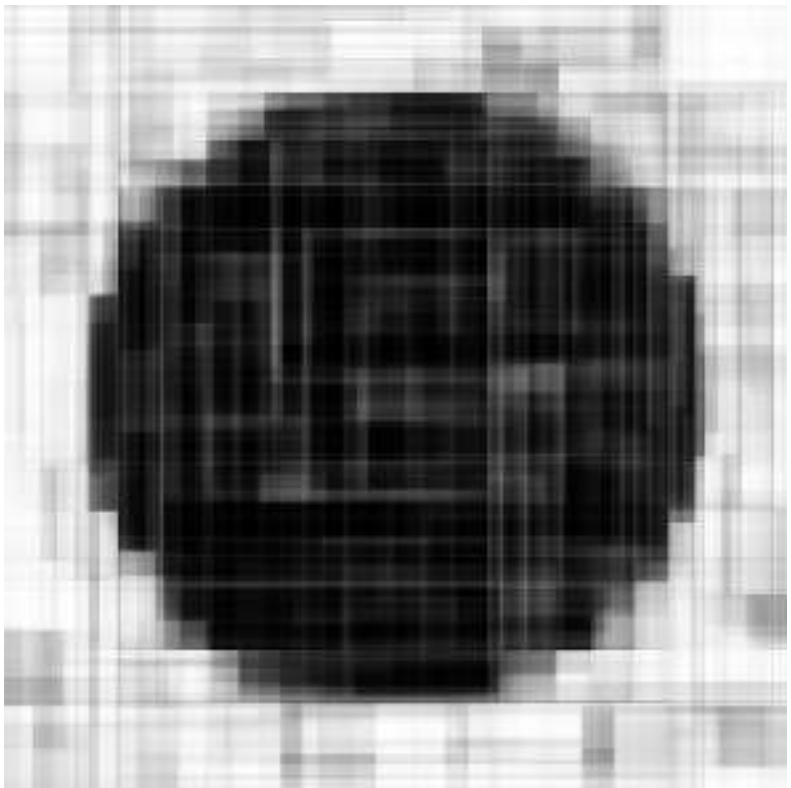
Probabilité
entre 0 et 1



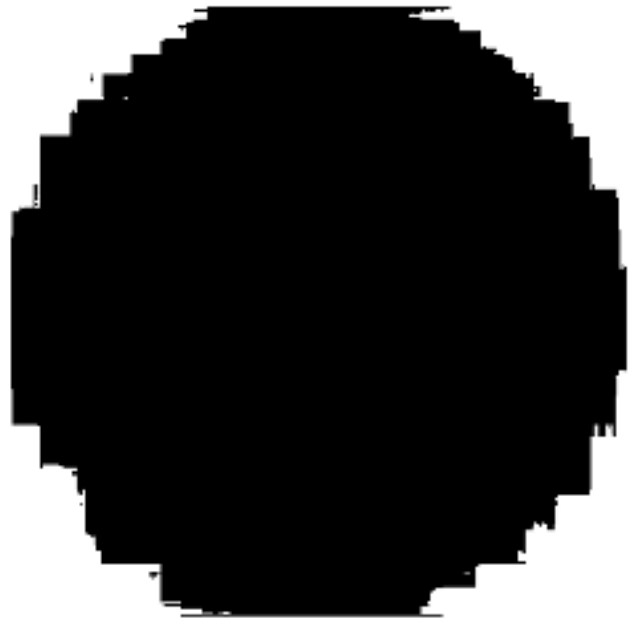
Prédiction
décision dure : classe 0 ou 1

Une forêt aléatoire (splits décidés sur seul 1 feature tiré au hasard)

`min_samples_leaf = 10, bootstrap = False, max_features = 1`



Probabilité
entre 0 et 1



Prédiction
décision dure : classe 0 ou 1

Combinaison séquentielle de prédicteurs (par boosting)

Une manière alternative de combiner plusieurs prédicteurs $f_1, \ldots, f_M$ consiste à les construire séquentiellement, afin d'obtenir finalement une combinaison linéaire performante

$$f = \sum_{m=1}^{M} \alpha_m f_m.$$

La stratégie la plus classique est le *Boosting*, qui est utilisable avec des prédicteurs qui peuvent être entraînés avec des exemples *pondérés*.

Algorithme AdaBoost

L'algorithme de Boosting le plus connu est *AdaBoost* (adaptive boosting), qui fonctionne de la manière suivante, pour une tâche de classification, avec les classes et les prédictions dans $\{-1, 1\}$:

1. Initialiser les pondérations des exemples de manière uniforme, c'est à dire, $w_n^1 = \frac{1}{N}, \ n = 1, \dots N$.
2. Pour $m = 1, \dots, M$
   - 2.1 entraîner f_m avec le jeu de données pondéré par $w_1^m, \dots, w_N^m$.
   - 2.2 calculer son erreur (pondérée) $e(f_m) = \sum_{n:f_m(x_n) \neq y_n} w_n^m$ (comprise entre 0 et 1)
   - 2.3 calculer la pondération qu'aura f_m dans la somme (=confiance en la prédiction) : $\alpha_m = \frac{1}{2} \log \frac{1-e(f_m)}{e(f_m)}$ (comprise entre $-\infty$ et $+\infty$)
   - 2.4 actualiser les poids des exemples pour le prochain classifieur $w_n^{m+1} \propto w_n^m \exp\left(-\alpha_m y_n f_m(x_n)\right)$ (plus de poids sur les exemples où le classifeur s'est trompé)
3. retourner le prédicteur $f = \sum_{m=1}^{M} \alpha_m f_m$.

Un classifieur fort peut être obtenu à partir de classifieurs faibles (si leur erreur de prédiction combinée est toujours inférieure 50%)

Un résultat théorique important est que, en notant f_w le prédicteur entraîné avec les poids $w_1, \ldots, w_n$, si

$$\forall w_1, \ldots, w_n \geq 0 \quad \text{avec} \quad \sum_n w_n = 1,$$

$$\exists \epsilon > 0 \quad \text{t.q.} \quad e(f_w) \leq \frac{1}{2} - \epsilon$$

alors AdaBoost peut construire un f qui a une erreur d'apprentissage égale à zéro.

AdaBoost peut être dérivée à partir d'une Loss exponentielle (et d'autre algorithmes de boosting pourraient être construits similairement à partir d'autres Loss)

Avec des classes binaires dans $\{-1, 1\}$, et des prédictions dans $\mathbb{R}$, si on considère l'erreur exponentielle

$$\mathscr{L}(f) = \sum_{n=1}^{N} \exp\left(-y_n f(x_n)\right)$$

AdaBoost peut être dérivé comme un algorithme qui ajoute à chaque itération le prédicteur qui réduit le plus $\mathscr{L}$ localement, de manière similaire à une descente de gradient.

On parle donc de *Gradient Boosting* qui peut être généralisé à d'autre fonctions de perte (entropie croisée, erreur quadratique).

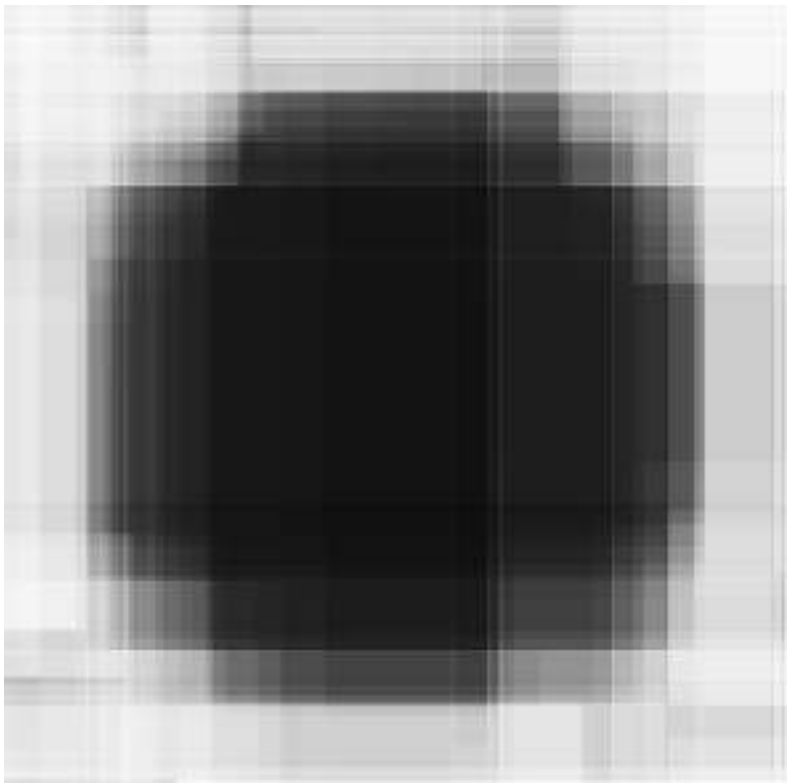
Exemple de gradient boosting dans scikit-learn

Gradient Boosting est disponible dans `scikit-learn`.

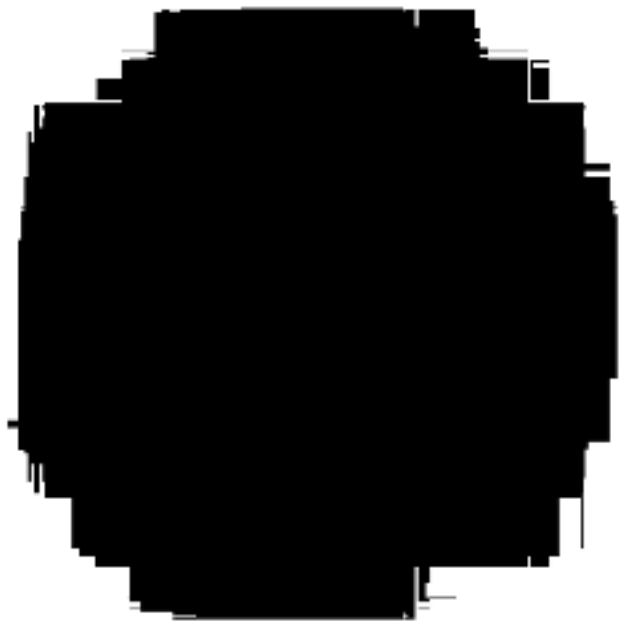
```
sklearn.ensemble.GradientBoostingClassifier(min_samples_leaf = min_samples_leaf)
```

Boosting de 100 arbres

`min_samples_leaf = 10`



Probabilité



Prédiction

Arbres de décision et méthodes ensemblistes : conclusions

Arbres de décision :

- formalisme flexible : capable de tenir compte d'une grande variété de type de variables (catégories, variables continues, discrètes)
- compatible avec structure euclidienne (mais elle n'est pas nécessaire)
- méthodes très utiles en pratique (performantes)
- méthodes d'apprentissage pour construire un arbre : méthode CART (Classification And Regression Tree)
- arbres construits récursivement : décision de split basée sur un critère
- critères pour régression (variance empirique) ou qualité des distribution crées dans les feuilles impureté de Gini, entropie de Shannon
- méthode de régularisation (élagage, critères d'arrêt)

Méthodes ensemblistes (ne s'appliquent pas qu'aux arbres) :

- méthodes parallèles (bagging)
  - bootstrapping (échantillonage aléatoire des données)
  - forêts aléatoires (randomisation dans l'arbre)
- méthodes séquentielles (boosting) : accent sur les exemples mal classifiés

Guide de lecture pour ce cours

Chloé-Agathe Azencott “Introduction au Machine Learning”,
Dunod, 2019, ISBN 978-210-080153-4
Chapitre 9 : Arbres et forêts

References

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