

Machine Learning

Cours 3

Jean-Claude Houbart

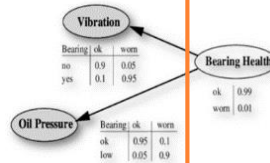
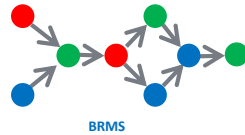
houbart.jc@gmail.com

Concepts préalables

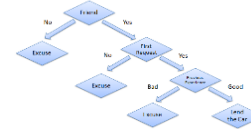
Jean-Claude Houbart

houbart.jc@gmail.com

Types de modèle IA



Réseaux Bayésiens



Arbres de décision



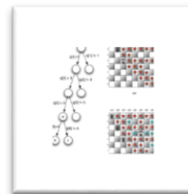
Modèles de forêt

Données

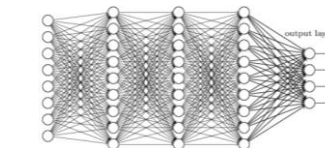
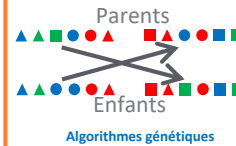
Machine Learning

Règles

Expertises



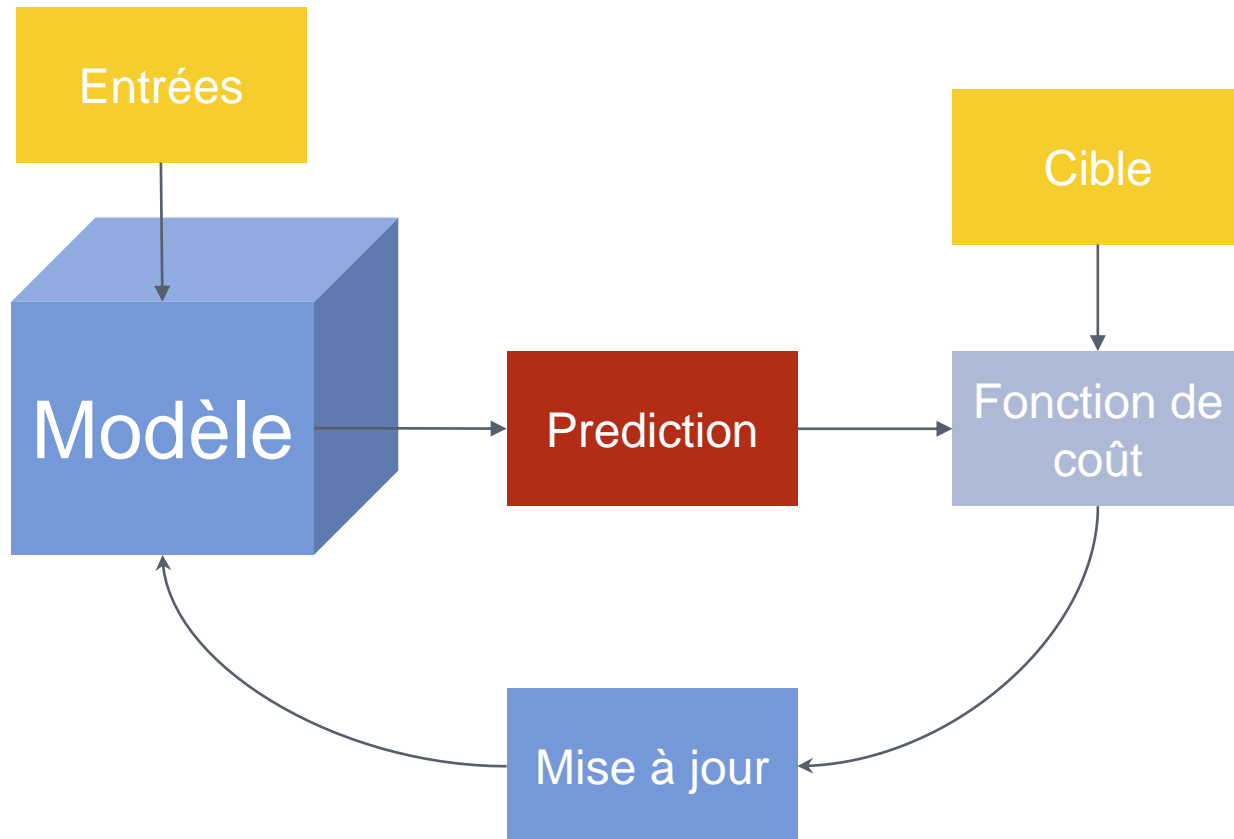
Programmation par contraintes



Réseaux de neurones

Boîte noire

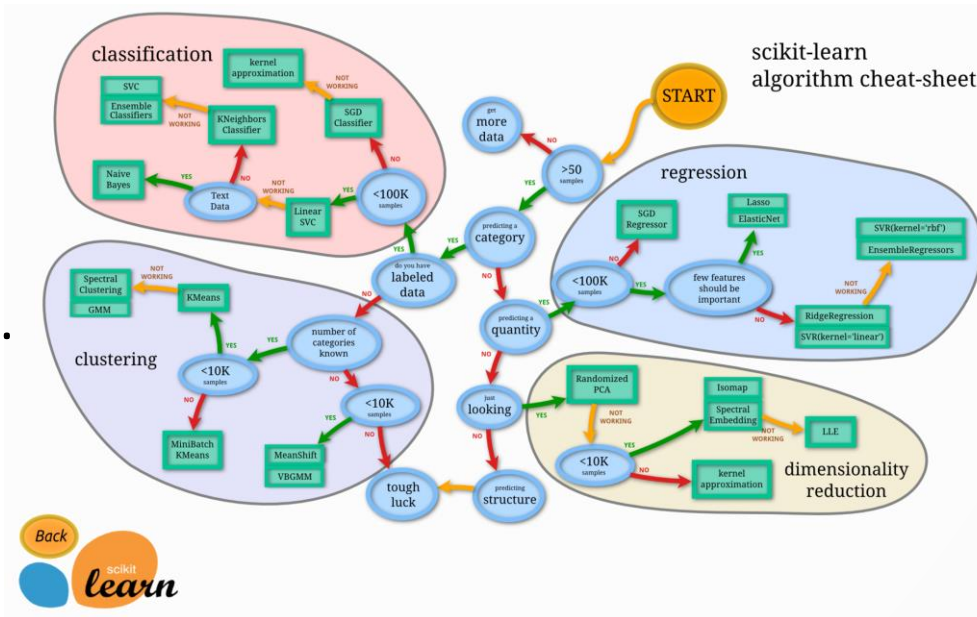
Apprentissage



Scikit Learn (sklearn)



- **Scikit-learn** est une boîte à outils Python pour le Machine Learning.
- Créé en France (INRIA), utilisé mondialement.
- Elle intègre de nombreux algorithmes IA, à l'exception des réseaux de neurones.
- API standardisée pour tous les algorithmes (objets **Classifier** et **Regressor**).
- Outils de préparations de données, combinaisons et évaluation de modèles.
- Ne fonctionne pas sur GPU.



Le Dataset Iris



Iris Versicolor



Iris Setosa



Iris Virginica

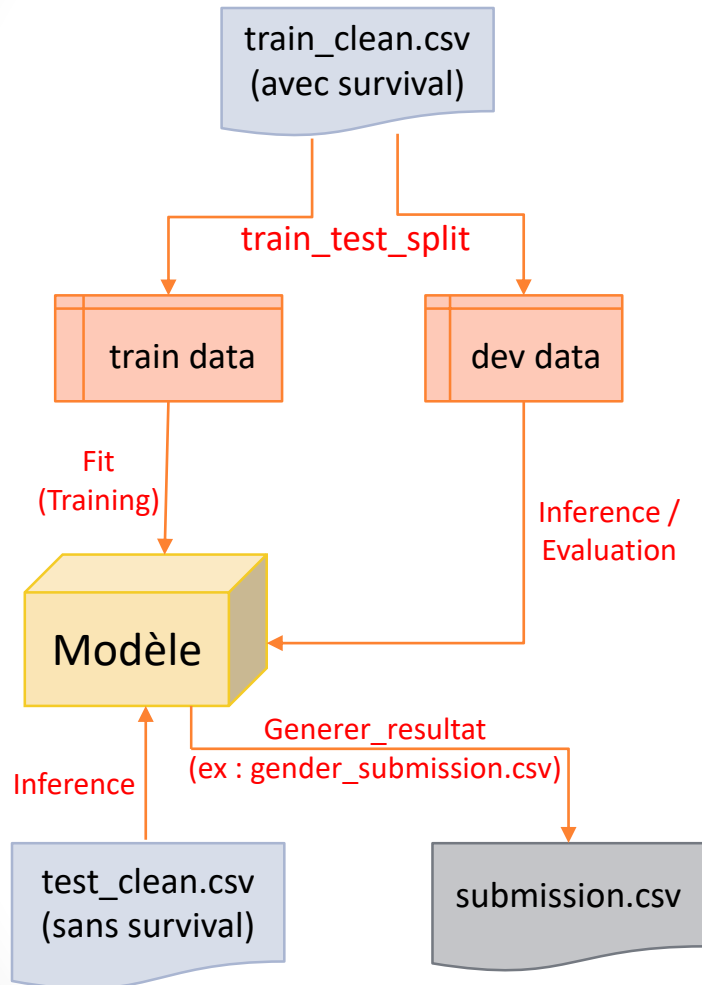
	sepal length	sepal width	petal length	petal width	target
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa

- 50 exemples pour chaque type d'Iris.
- Disponible dans Scikit_Learn en utilisant la fonction `load_iris()` du package [sklearn.datasets](https://scikit-learn.org/stable/datasets/iris.html).
- Pour les besoins de l'exemple, des valeurs manquantes ont été ajoutées.

Projet Titanic sur Kaggle

- **Kaggle** (<https://www.kaggle.com/>) est une plateforme web de compétitions en science des données. Propose aussi des cours, des exemples et des datasets. Racheté par Google en 2017.
- Connexion avec votre compte Google.
- Aller dans Competitions puis chercher « **Titanic - Machine Learning from Disaster** »
- Joindre la compétition.
- Vous pourrez alors envoyer vos prédictions, et connaître votre classement. A partir de **78%** de bonne réponses, vous êtes bien.

Comprendre les 3 datasets



- On utilise les données **train_clean.csv** qui contient la réponse pour entraîner le modèle et mesurer sa performance.
- Quand le modèle est prêt, on génère le fichier **submission.csv** depuis le fichier **test_clean.csv**.

→ kaggle

Arbres de Décision

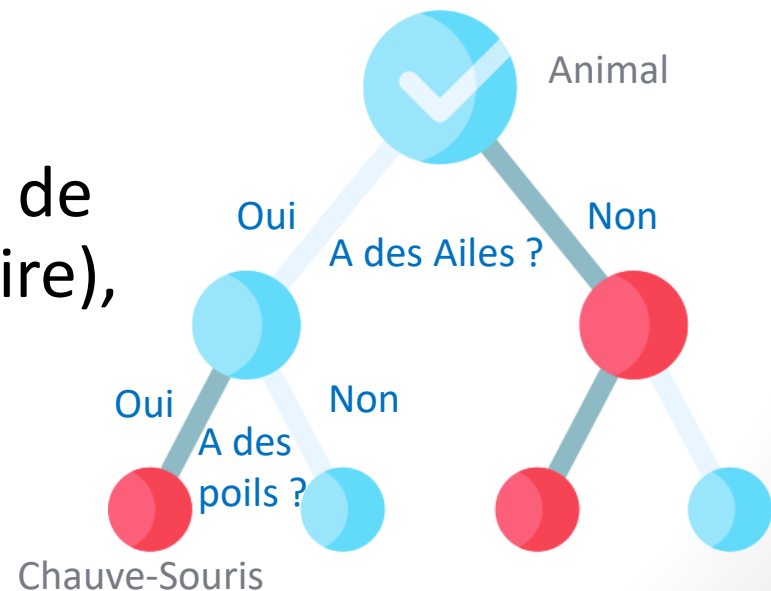
Jean-Claude Houbart

houbart.jc@gmail.com



Définition

- Un arbre de décision permet de classer des enregistrements par division hiérarchiques en sous-classes
 - un nœud représente un sous-ensemble depuis la racine
 - un arc représente une règle de partitionnement de la classe source
- Un attribut sert d'étiquette de classe (attribut cible à prédire), les autres permettant de partitionner




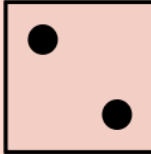
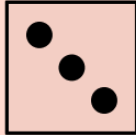


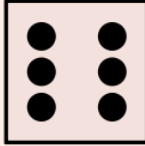


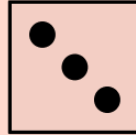
Exemple de construction d'arbre

```
## Chargement du Dataset Iris##  
iris = load_iris()  
iris.feature_names=['Longueur Sépale (cm)', 'Largeur Sépale  
(cm)', 'Longueur Pétale (cm)', 'Largeur Pétale (cm)']  
  
X = iris.data[:, 2:] # petal length and width  
y = iris.target  
  
## Apprentissage de l'arbre de décision ##  
tree_clf = DecisionTreeClassifier(max_depth=2,  
                                   random_state=42)  
tree_clf.fit(X, y)
```

Voir cours3_ex1_constructionArbre.ipynb

Random_state

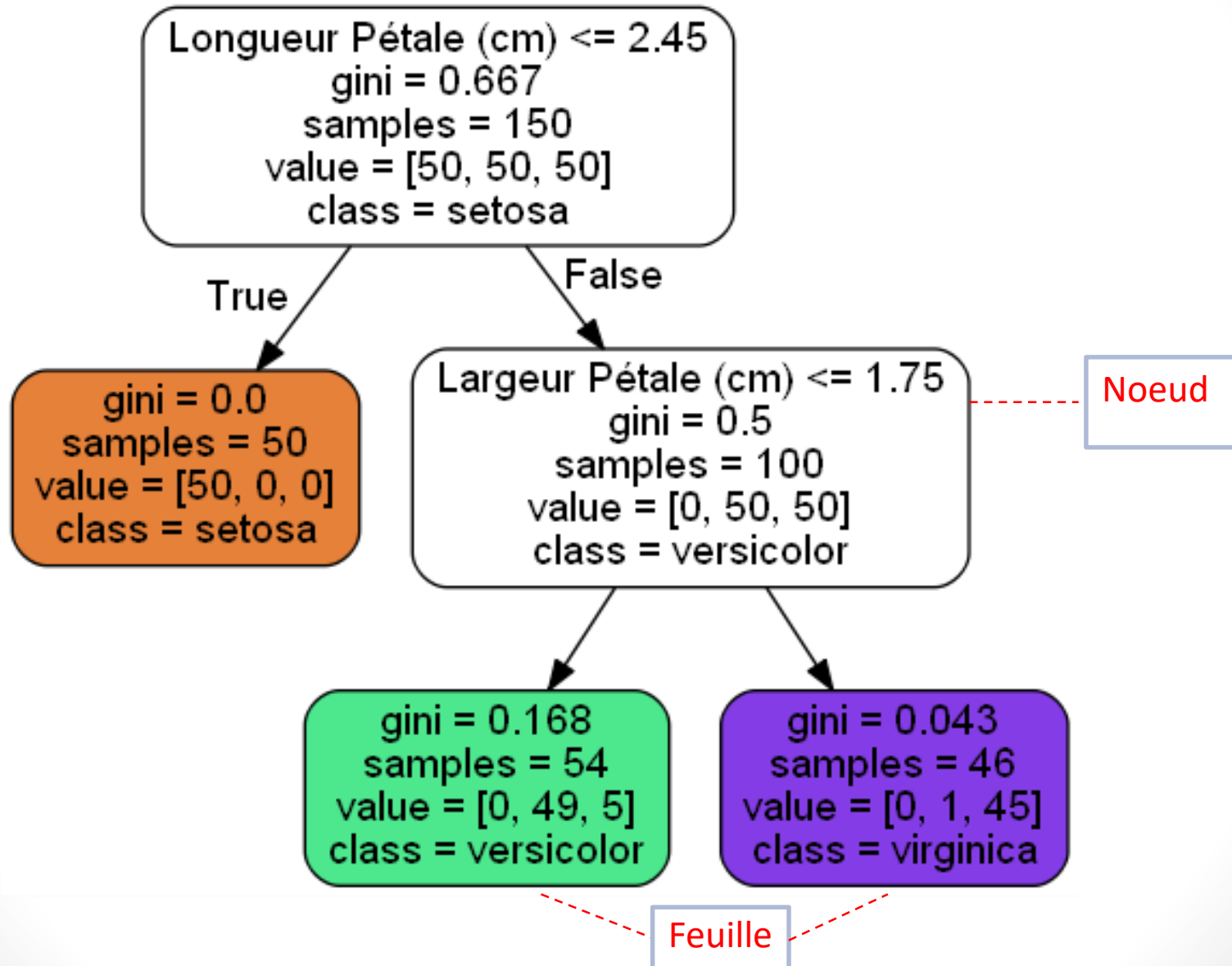
- Permet d'obtenir toujours le même résultat d'un tirage aléatoire.

	Tirage 1	Tirage 2	Tirage 3
Pas de Random_state :			
Random_state = 1 :			
Random_state = 2 :			

- Utile pour comparer plusieurs exécution sans modification des paramètres aléatoires.



Un arbre de décision





Détail d'un nœud

Condition de partitionnement

Pour créer les nœuds en-dessous

Longueur Pétale (cm) ≤ 2.45

gini = 0.667

samples = 150

Value = [50, 50, 50]

Class=setosa

« Pureté »

Mesure d'homogénéité
Doit être maximisée
Et donc Gini minimisé

Population totale

Classe la plus probable

Ici équiprobables donc la première

Population par classe

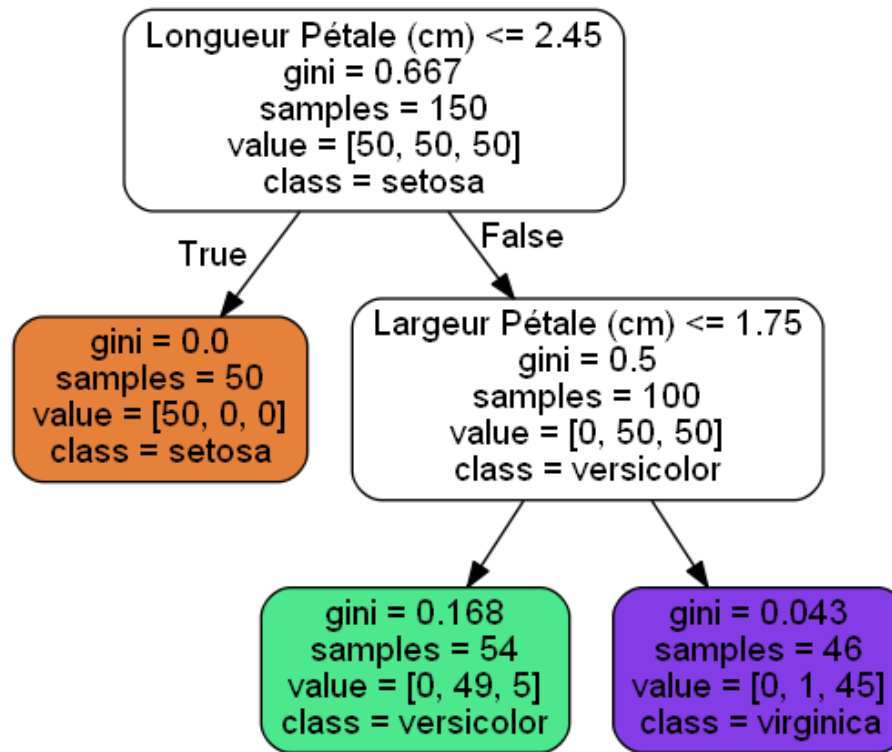
[setosa, versicolor, virginica]

0

1

2

Détail d'un nœud



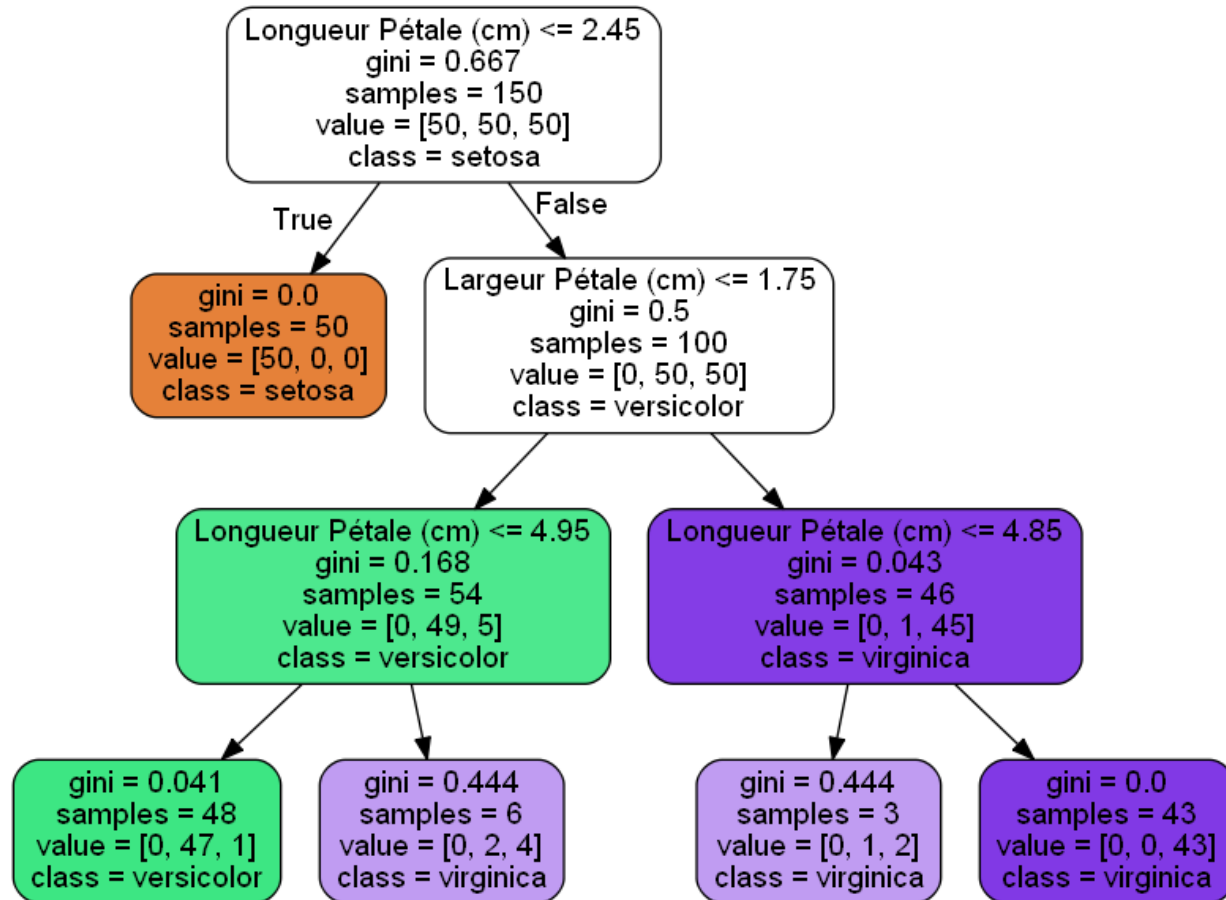
```
ma_nouvelle_fleur=[2,2,5, 1.5]
print('Probabilités :', np.round(tree_clf.predict_proba([ma_nouvelle_fleur]),2))
print('N° classe :', tree_clf.predict([ma_nouvelle_fleur]))
print('Classe :', iris.target_names[tree_clf.predict([ma_nouvelle_fleur])])
```

Probabilités : [[0. 0.91 0.09]]

N° classe : [1]

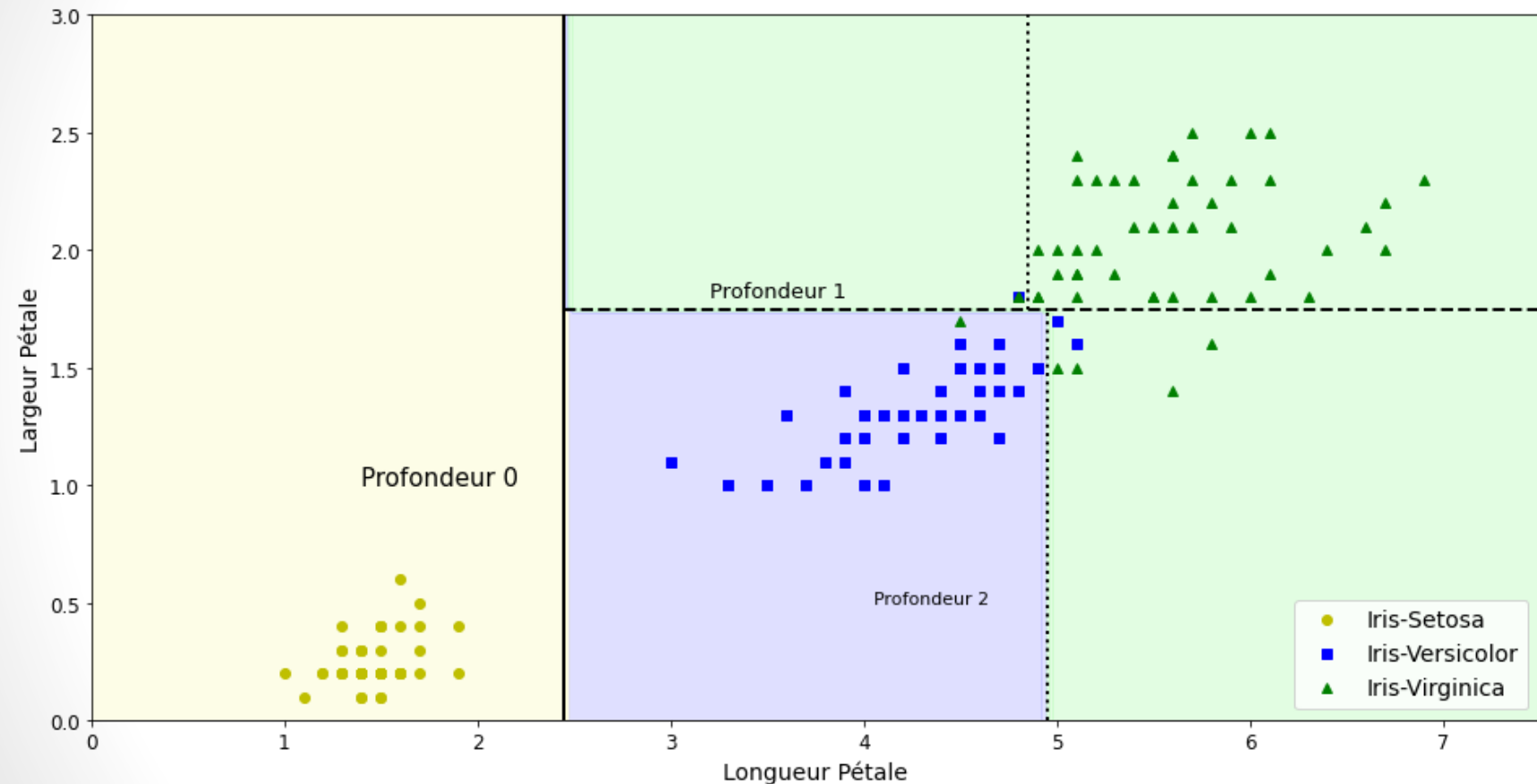
Classe : ['versicolor']

Visualisation d'un arbre à 3 niveaux



Voir cours3_ex2_Visualisation.ipynb

Visualisation du modèle (3 niveaux)





Calcul de la Pureté par formule de gini

- La méthode principale est la formule de **gini**.
- Si tous les membres de l'apprentissage sont la même classe, la valeur gini est 0.
- La formule générale pour le nœud i est :

$$G_i = 1 - \sum_{k=1}^n p_{i,k}^2$$

... avec $p_{i,k}$ la proportion de la classe k dans le nœud i .

- Exemple pour le nœud « versicolor »

gini = 0,168
samples = 54
Value = [0, 49, 5]
Class=versicolor

$$\begin{aligned} \text{gini} &= 1 - [(0/54)^2 + (49/54)^2 + (5/54)^2] \\ &= 1 - [0 + 0,9074^2 + 0,0926^2] = 0,168 \end{aligned}$$



Calcul de la Pureté par l'entropie

- La formule générale pour l'entropie du nœud i est :

$$G_i = - \sum_{k=1}^n p_{i,k} \ln(p_{i,k})$$

... avec toujours $p_{i,k}$ la proportion de la classe k dans le nœud i .

- Exemple pour le nœud « versicolor »

entropy= 0,308

samples = 54

Value = [0, 49, 5]

Class=versicolor

$$\begin{aligned} \text{gini} &= - (0/54)\ln(0/54) - (49/54)\ln(49/54) - (5/54)\ln(5/54) \\ &= - 0 - 0,9074\ln(0,9074) + 0,0926\ln(0,0926) \\ &= - 0,0383 - 0,0039 = 0,308 \end{aligned}$$

- La plupart de temps les deux formules sont très similaires. Cependant le gini est un peu plus rapide à calculer alors que l'entropie produit des arbres légèrement mieux équilibrées (la population est mieux répartie entre les noeuds).

Algorithme CART

- L'algorithme « **Classification And Regression Tree** » permet de construire et entraîner un arbre de décision.
- Pour un nœud, il cherche une feature et un seuil (par exemple Longueur Pétale $\leq 2,45$) qui minimise la **pureté pondérée des nœuds inférieurs 1 & 2**.

$$\text{Coût} = \frac{\text{Nb}_{nd1}}{\text{Nb}_{total}} \text{Pureté}_{nd1} + \frac{\text{Nb}_{nd2}}{\text{Nb}_{total}} \text{Pureté}_{nd2}$$

- Ensuite On recommence sur les nœuds ainsi générés.
- L'hyperparamètre **max_features** permet de limiter le nombre de features testées à chaque nœud.
- La solution ainsi trouvée n'est **pas forcément optimale**, mais généralement suffisamment bonne.
- Il existe de nombreux autres algorithmes.



Arrêt de l'algorithme

- L'algorithme CART s'arrête s'il ne peut pas retrouver de division qui augmente la pureté. La somme pondérée des deux nœuds générés est supérieure à celle du nœud parent.
- Le plus souvent, on va utiliser un hyperparamètre comme condition d'arrêt :
 - `max_depth` : Profondeur maximale de l'arbre.
 - `min_samples_split` : Nombre minimum d'échantillons qu'un nœud doit avoir pour être divisé.
 - `min_samples_leaf` : Nombre minimum d'échantillons qu'une feuille (leaf) doit avoir pour être créée.
 - `min_weight_fraction_leaf` : Fraction minimum du total qu'une feuille doit avoir.
 - `max_leaf_nodes` : nombre maximum de feuilles.
- Sans condition d'arrêt, le sur-apprentissage est quasi-certain.



Spécificités des arbres de décision

- L'arbre de décision est **explicable** (contrairement à l'ANN ou au modèle de forêt).
- Il n'est **pas nécessaire de normaliser** les données (contrairement à l'ANN)
- Capable de traiter des problèmes de **régression**
- **Extrêmement rapide** en Inférence.
- **Instable** : Une petite modification des données entrantes peut amener à un arbre totalement différent.
- Le nombre de paramètres n'est pas défini à l'avance (modèle non paramétrique) => **Fort risque de sur-apprentissage**.

Random Forest

Jean-Claude Houbart

houbart.jc@gmail.com



Spécificité du Random Forest

- En première approximation, un modèle **Random Forest** est un Bagging d'arbres de décisions, dont chaque échantillon a la taille du jeu de données initial (mais n'est pas identique !).
- En plus, il n'utilise qu'un **sous-ensemble aléatoire des features** pour chaque arbre.
- Dans le modèle **extra-Tree** (*Extremely Randomized Trees*), on met en plus des seuils aléatoires.
- Sklearn propose deux estimateurs optimisée pour le Random Forest: **RandomForestClassifier** et **RandomForestRegressor**.
- Les paramètres sont globalement ceux des arbres de décision **Plus** ceux du Bagging.
- Le paramètre **feature_importances_** est un bon moyen de connaître l'importance relative des features.

Exemple de Modèle de Forêt

```
# Importation des librairies
```

```
from sklearn.datasets import load_iris
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
# Importance des Features Iris Data Set
```

```
iris = load_iris()
```

```
iris.feature_names=['Longueur Sépale (cm)', 'Largeur Sépale (cm)',  
                   'Longueur Pétale (cm)', 'Largeur Pétale (cm)']
```

```
rnd_clf = RandomForestClassifier(n_estimators=500,  
                                n_jobs=-1, random_state=42)
```

```
rnd_clf.fit(iris["data"], iris["target"])
```

```
for name, score in zip(iris["feature_names"],  
rnd_clf.feature_importances_):
```

```
print('%s: %i%%' %(name, int(score*100)))
```

Longueur Sépale (cm): 11%

Largeur Sépale (cm): 2%

Longueur Pétale (cm): 44%

Largeur Pétale (cm): 42%

Voir cours3_ex3_ForestModel.ipynb