# Cours de Data Science

Marc Tommasi

25 octobre 2022

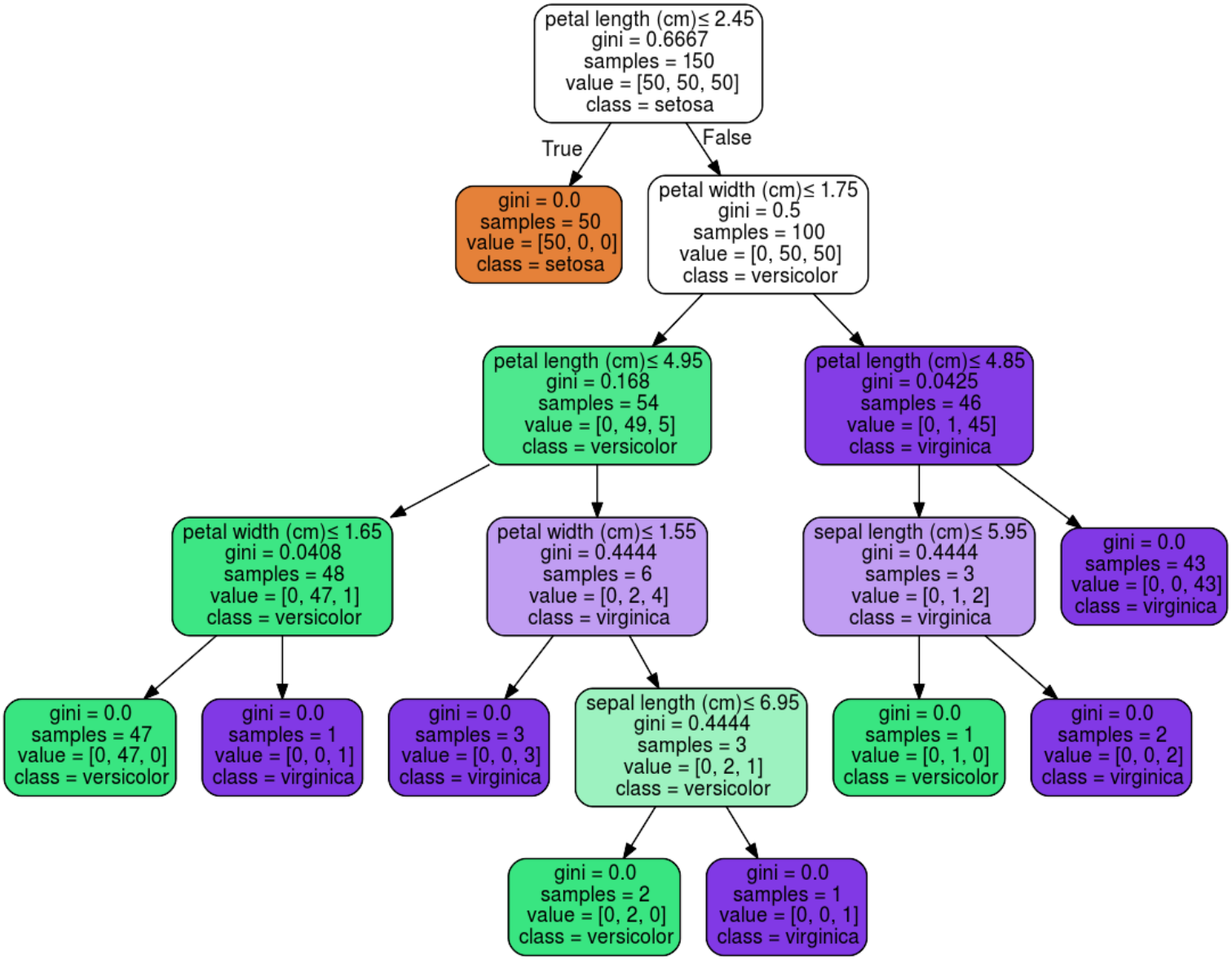
# Outline

# Outline

# Principes

- Classifieur de $\mathcal{X}$ dans $\mathcal{Y}$
- Modèle sous forme d'un ensemble de règles de décision successives, représentées dans un arbre
- Modèle simple à interpréter quand l'arbre est petit
- Exemple sur le jeu de données iris, 3 classes : setosa, virginica, versicolor ; 150 exemples ; attributs sepal length, sepal width, petal length, petal width.

# Exemple


petal length (cm)≤ 2.45
gini = 0.6667
samples = 150
value = [50, 50, 50]
class = setosa
True
False
gini = 0.0
samples = 50
value = [50, 0, 0]
class = setosa
petal width (cm)≤ 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
class = versicolor
petal length (cm)≤ 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor
petal length (cm)≤ 4.85
gini = 0.0425
samples = 46
value = [0, 1, 45]
class = virginica
petal width (cm)≤ 1.65
gini = 0.0408
samples = 48
value = [0, 47, 1]
class = versicolor
petal width (cm)≤ 1.55
gini = 0.4444
samples = 6
value = [0, 2, 4]
class = virginica
sepal length (cm)≤ 5.95
gini = 0.4444
samples = 3
value = [0, 1, 2]
class = virginica
gini = 0.0
samples = 43
value = [0, 0, 43]
class = virginica
gini = 0.0
samples = 47
value = [0, 47, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
sepal length (cm)≤ 6.95
gini = 0.4444
samples = 3
value = [0, 2, 1]
class = versicolor
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
gini = 0.0
samples = 2
value = [0, 0, 2]
class = virginica
gini = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica

# Fonctionnement

- On part de la racine, avec un exemple (e.g. 3, 2, 3, 2)
- On passe à travers les tests, chaque test coupe l'espace en 2 parties.
- On désigne donc un partitionnement récursif de l'espace de description
- Chaque feuille donne une étiquette à une partie.
- Bonne visualisation dans le livre de Jake VandenPlas.

# Algorithmes

- C'est une méthode qui introduit deux biais : choix de la classe de fonctions + biais algorithmique
- Le biais algorithmique provient d'une heuristique gourmande :
  - on construit l'arbre de la racine aux feuilles,
  - à chaque étape on développe un noeud correspondant à une partie des données
  - on sélectionne le meilleur test selon un critère de gain
  - on ne remet plus en cause ce choix
- Il existe plusieurs algorithmes : ID3, C4.5, C5, CART,...
- Sklearn implante CART

# Explications avec ID3

- Les attributs $A = \{x_0, x_1, \ldots, x_p\}$ sont tous binaires

```
def id3(S,A):
  """ S : echantillon, A: attributs """
  if tous les éléments de S sont de même classe ou
     A est vide:
    retourner une feuille contenant la classe majoritaire

  Soit j l'attribut qui maximise le gain
    t_l = id3({(x,y) de S tq x_j=0}, A\{j})
    t_r = id3({(x,y) de S tq x_j=1}, A\{j})
    retourner l'arbre de noeud qui teste x_j=1
              avec les fils t_l (cas False) et t_r (cas True).
```

# Fonctions de gain

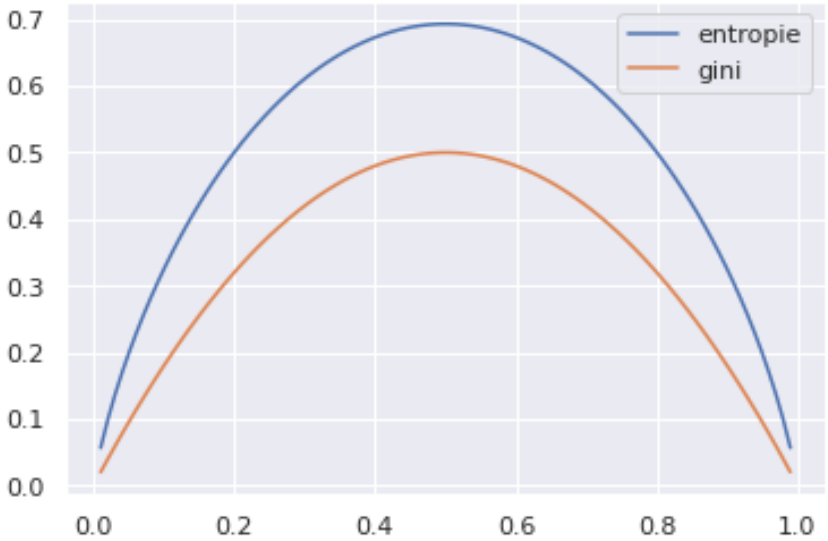
- Gain : différence observée entre absence et présence du test dans l'arbre.
- Notation : $\mathbb{P}_S[F]$ est la probabilité de $F$ quand on tire uniformément dans $S$
  - (si le noeud a $m$ exemples et $m_l$ passent à gauche et $m_r$ passent à droite avec un test $x_i$ alors $\mathbb{P}_S[x_i = 1] = m_r/m$)
- le gain sera calculé avec une fonction $C$ à définir :

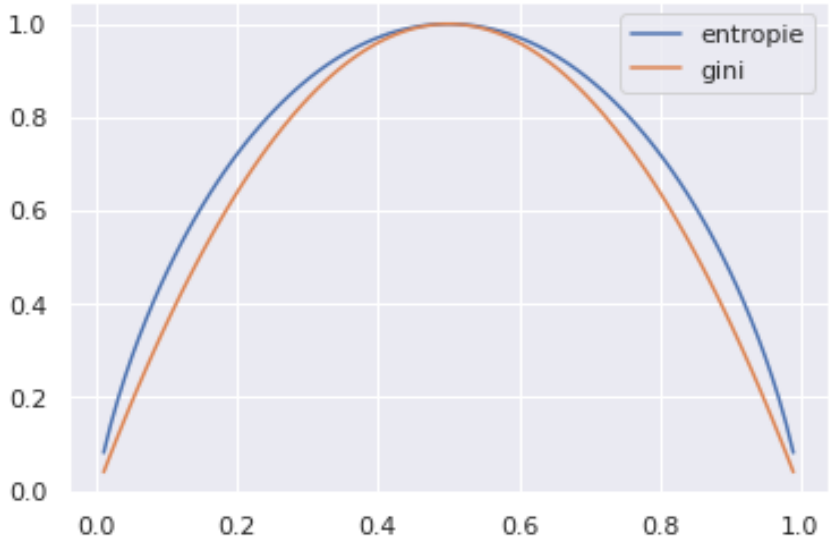
$$\text{Gain}(S, i) = C(\mathbb{P}_S[y = 1]) - \big(\mathbb{P}_S[x_i = 1]C(\mathbb{P}_S[y = 1 \mid x_i = 1]) + \mathbb{P}_S[x_i = 0]C(\mathbb{P}_S[y = 1 \mid x_i = 0])\big)$$

- Erreur d'apprentissage : différence entre les erreurs faites par l'arbre avant et après l'introduction de ce noeud. $C(a) = \min(a, 1 - a)$
- Gain en information : différence entre l'entropie avant et après le test. $C(a) = -a\log(a) - (1 - a)\log(1 - a)$
- Gini, pureté : $C(a) = 2a(1 - a)$ (facteur 2 pour avoir un maximum à 1)

# Entropie et gini

- Vont favoriser les tests qui réalisent la meilleure séparation : $\mathbb{P}_S[y \mid x_i]$ proche de 0 ou de 1.




- $-a\log(a) - (1-a)\log(1-a)$ et $2a(1-a)$
- $-a\log_2(a) - (1-a)\log_2(1-a)$ et $4a(1-a)$

# Limiter l'overfitting

- Compromis biais/complexité : plus la profondeur est grande, plus la classe de fonctions est complexe, plus le biais est faible mais plus les arbres de décision auront tendance à l'overfitting.
- borne sur la profondeur (comme dans sklearn, `max_depth`)
- l'élagage (pruning) consiste à supprimer des branches : remplacer un noeud par une étiquette de classe
  - Approche bottom-up avec un test statistique : ($\xi^2$ ou évaluation de l'erreur).

# Le cas des attributs continus

- discrétisation considérant tous les seuils possibles observés sur l'échantillon d'apprentissage
- le calcul pour ces $m$ tests possibles pourrait être $O(dm^2)$ mais peut être réduit à $O(dm\log(m))$.

# Arbres de régression

- on fait la moyenne des exemples qui arrivent dans une feuille pour déterminer la valeur à prédire
- la fonction de coût pour la construction utilisée est par exemple MSE

# Avantages et inconvénients

- lisibilité du modèle
- complexité algorithmique élevée pour trouver le meilleur arbre, mais approche heuristique rapide.
- difficulté de régler la profondeur, les critères qui évitent le sur-apprentissage