Introduction à l'apprentissage automatique

CSI 4506 - Automne 2026

Marcel Turcotte
Version: sept. 13, 2026 17h01

Préambule

Message du jour (1/2)

The AI Researcher Who Just Quit Anthropic Says It's 'Crunch Time for Humanity', *Wired*, 2026-09-09.

- Anthropic affirme avoir déjoué des tentatives de fabrication d'armes biologiques, La Presse, 2026-09-11.
- L'intelligence artificielle pourrait-elle vraiment anéantir l'humanité ?, Bruno Guglielminetti, Mon Carnet, 2026-09-11.

Message du jour (2/2)

Why Do We Tell Ourselves Scary Stories About AI?, *Quanta Magazine*, 2026-04-10.

Rappel du cours précédent

- Les méthodes **symboliques** et **connexionnistes** constituent deux grandes traditions de l'IA.
- Définir l'**intelligence** demeure difficile.
- Russell et Norvig décrivent **quatre perspectives sur l'IA** : penser comme un humain, agir comme un humain, penser rationnellement et agir rationnellement.
- L'**IA étroite** vise des tâches particulières; l'**intelligence artificielle générale** constitue un objectif plus vaste et controversé.
- Des questions difficiles et non résolues concernent notamment l'**incarnation**, l'**agentivité**, la **sensibilité**, la **conscience**, les **émotions** et la **cognition**.

Objectifs d'apprentissage

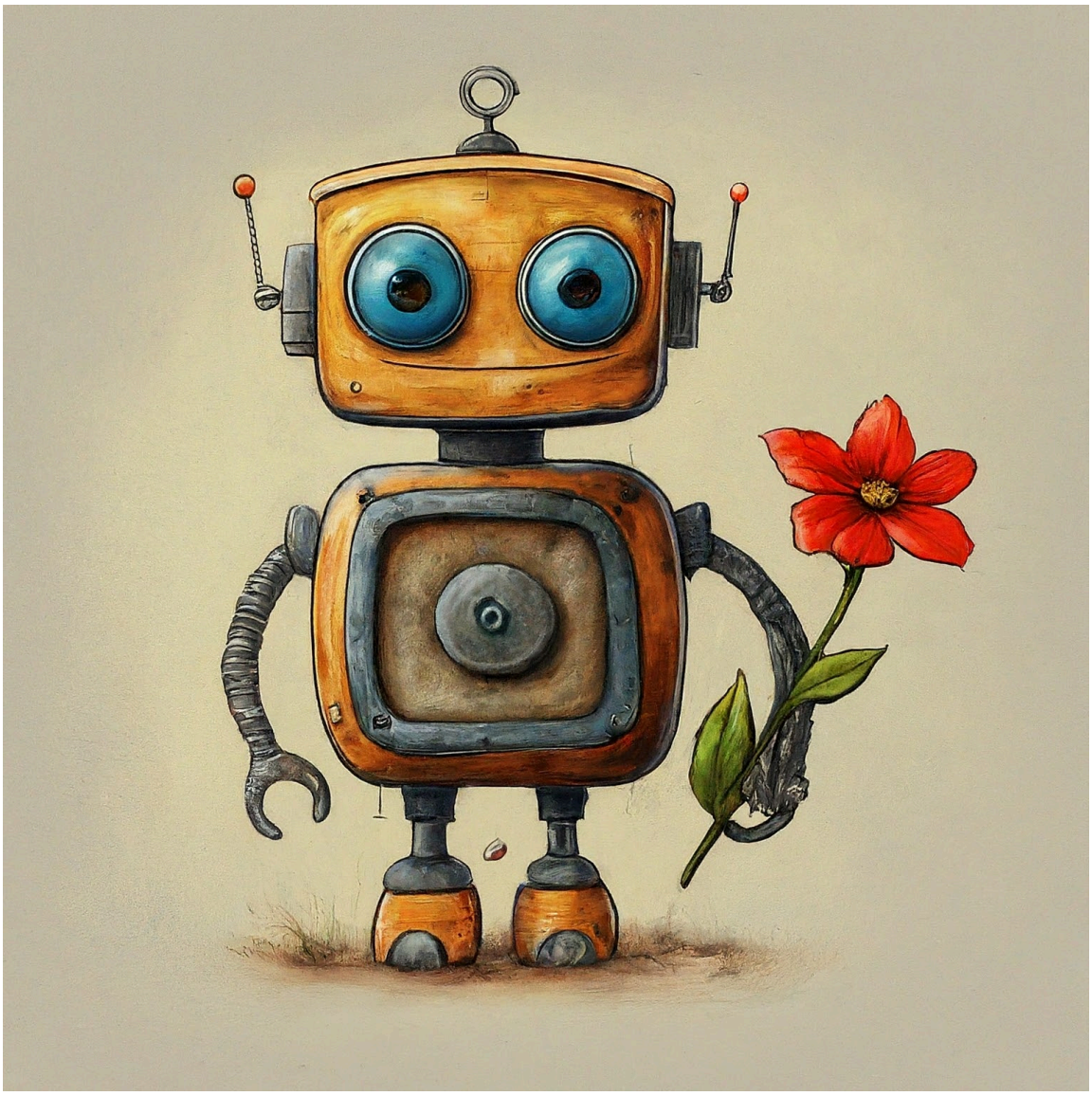
- **Distinguer** l'apprentissage supervisé, non supervisé et par renforcement selon la rétroaction disponible
- **Identifier** les exemples, les attributs et les cibles dans un jeu de données supervisé
- **Différencier** la classification de la régression, ainsi que l'apprentissage de l'inférence
- **Appliquer** le flux de travail élémentaire de scikit-learn : charger et explorer des données, ajuster un modèle et effectuer des prédictions
- **Expliquer** pourquoi un jeu de test distinct est nécessaire pour estimer la performance sur des données inédites

Lecture : Russell et Norvig (2020), chapitre 19, « Apprentissage à partir d'exemples ».

Introduction

Justification

Pourquoi un programme informatique devrait-il *apprendre* ?



Attribution: Gemini 1.5 Flash, 14 août, 2024, avec l'invite: "In the style of a children's book, create the image of a cute robot holding a red flower."

1. **Adaptabilité et amélioration continue :**

- **Adaptabilité aux environnements dynamiques :** Les programmes qui apprennent peuvent s'adapter aux conditions changeantes, assurant ainsi un fonctionnement efficace dans des environnements dynamiques.

    - *Exemple :* Les voitures autonomes s'ajustant aux changements de trafic et de météo.
- **Amélioration continue :** L'apprentissage permet aux systèmes de rester à jour avec les dernières données et tendances sans intervention humaine.
    - *Exemple :* Les filtres anti-spam évoluant pour contrer les nouvelles techniques de spam.

1. **Performance et efficacité améliorées :**

- **Performance améliorée :** L'apprentissage permet aux programmes d'améliorer leurs performances en se basant sur des expériences passées ou des données.
    - *Exemple :* Les systèmes de recommandation affinant leurs suggestions avec plus de données utilisateurs.
- **Rentabilité :** L'automatisation du processus d'apprentissage réduit le besoin de mises à jour manuelles, entraînant des économies.
    - *Exemple :* Les systèmes de maintenance prédictive minimisant les inspections manuelles.

1. **Résolution de problèmes complexes et découverte de motifs cachés :**

- **Gestion des problèmes complexes :** Les algorithmes d'apprentissage abordent des problèmes trop complexes pour les systèmes statiques basés sur des règles.
    - *Exemple :* La reconnaissance d'images distinguant différents objets.
- **Découverte de motifs cachés :** Les modèles d'apprentissage peuvent découvrir des relations cachées dans les données qui ne sont pas évidentes pour les analystes humains.
    - *Exemple :* L'identification de relations génomiques complexes en bioinformatique.

1. **Personnalisation et évolutivité :**

- **Personnalisation :** L'apprentissage permet aux programmes de fournir des résultats adaptés aux utilisateurs individuels, améliorant ainsi l'expérience utilisateur.
    - *Exemple :* Les assistants virtuels apprenant les préférences des utilisateurs.
- **Évolutivité :** Les algorithmes d'apprentissage gèrent et analysent efficacement de grands ensembles de données, améliorant leur utilité.
    - *Exemple :* Les moteurs de recherche optimisant la pertinence des résultats grâce au machine learning.

1. **Innovation et recherche :**

- **Favoriser l'innovation :** Les algorithmes d'apprentissage peuvent simuler de nouvelles idées, conduisant à des avancées et découvertes.
    - *Exemple :* Les modèles d'apprentissage profond prédisant l'efficacité des médicaments en recherche pharmaceutique.

Définition

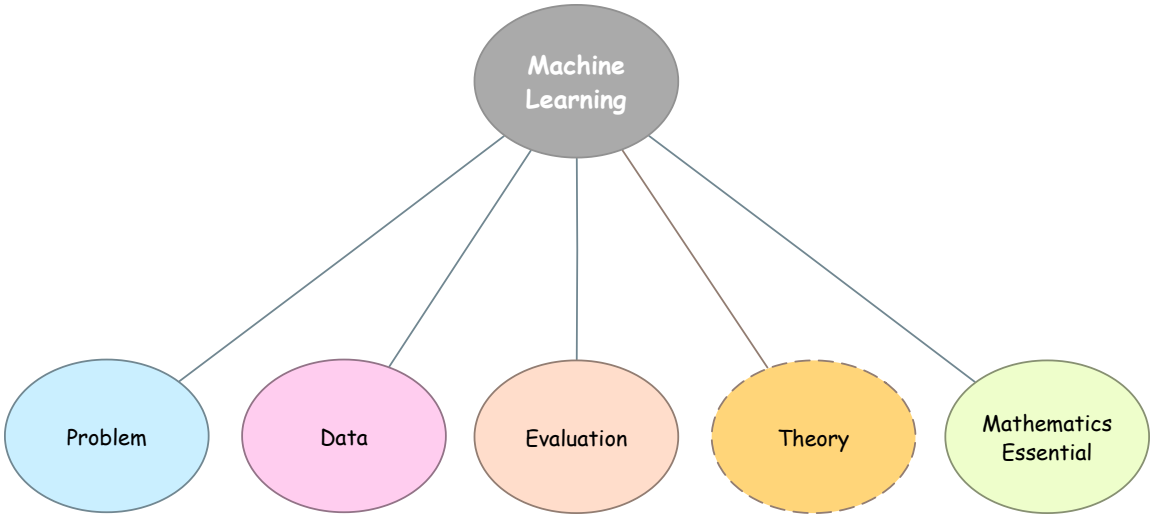
> **Mitchell (1997), page 2**
>
> Un programme informatique **apprend** à partir de **l'expérience** E par rapport à une certaine classe de **tâches** T et une **mesure de performance** P, si sa performance pour les tâches dans T, mesurée par P, s'améliore avec l'expérience E.

Tom M Mitchell. Machine Learning. McGraw-Hill, New York, 1997. (PDF)

Bien que ce livre ait été publié en 1997, il a exercé une influence considérable et reste pertinent. Contrairement à de nombreux autres livres sur l'apprentissage automatique, il est particulièrement captivant.

Cette définition particulière de l'apprentissage est souvent réutilisée.

Concepts



Voir: images/svg/ml_concepts-00.svg

Types de problèmes

Nous commencerons par trois grands cadres d'apprentissage, distingués selon la **rétroaction** disponible :

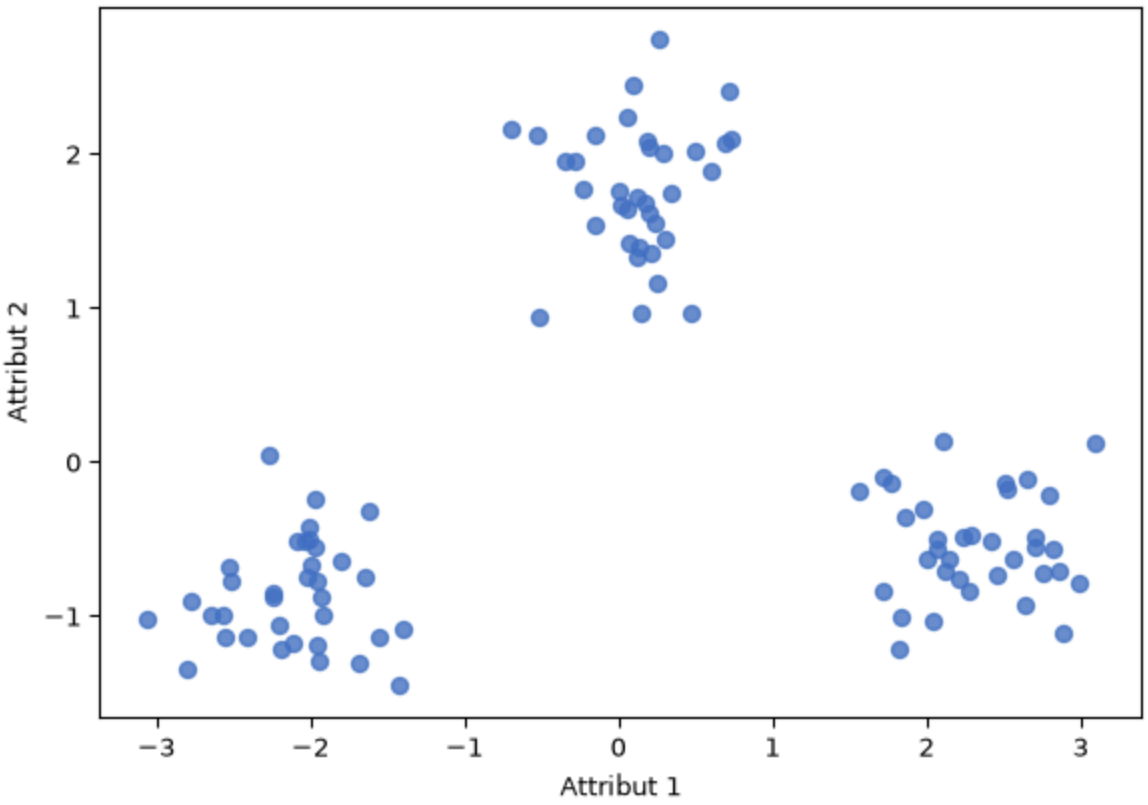
1. **Apprentissage non supervisé :** Les exemples n'ont pas d'étiquettes cibles.
2. **Apprentissage supervisé :** Chaque exemple d'entraînement est accompagné d'une cible.
3. **Apprentissage par renforcement :** Un agent reçoit des récompenses numériques en interagissant avec un environnement.
4. . .

L'apprentissage supervisé est le cadre le plus étudié dans ce cours et sans doute le plus intuitif pour commencer.

Apprentissage non supervisé

Données : des vecteurs d'attributs $x_1, x_2, \ldots, x_N$, mais **aucune étiquette cible** y_i.

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np

rng = np.random.default_rng(7)
centres = [(-2.0, -0.8), (0.2, 1.8), (2.3, -0.5)]
points = np.vstack([
    rng.normal(loc=centre, scale=0.42, size=(35, 2))
    for centre in centres
])

plt.scatter(points[:, 0], points[:, 1], color="#4472C4", alpha=0.8)
plt.xlabel("Attribut 1")
plt.ylabel("Attribut 2")
plt.gca().set_aspect("equal", adjustable="box")
plt.show()
```



Découvrir une **structure** sans étiquettes

Tous les points ont la même couleur puisqu'aucune étiquette de classe n'a été fournie. Un algorithme de regroupement tente d'inférer des groupes à partir des valeurs des attributs.

En **apprentissage non supervisé**, l'algorithme reçoit des exemples sans étiquettes cibles et tente d'y découvrir une structure utile.

Le **regroupement** (*clustering*) rassemble des exemples semblables selon les attributs et la mesure de distance choisis. Les groupes sont inférés plutôt que fournis par un enseignant. Ils ne doivent pas être interprétés automatiquement comme des catégories objectivement vraies ou pertinentes : le résultat peut changer selon les attributs, leur mise à l'échelle, l'algorithme et ses paramètres.

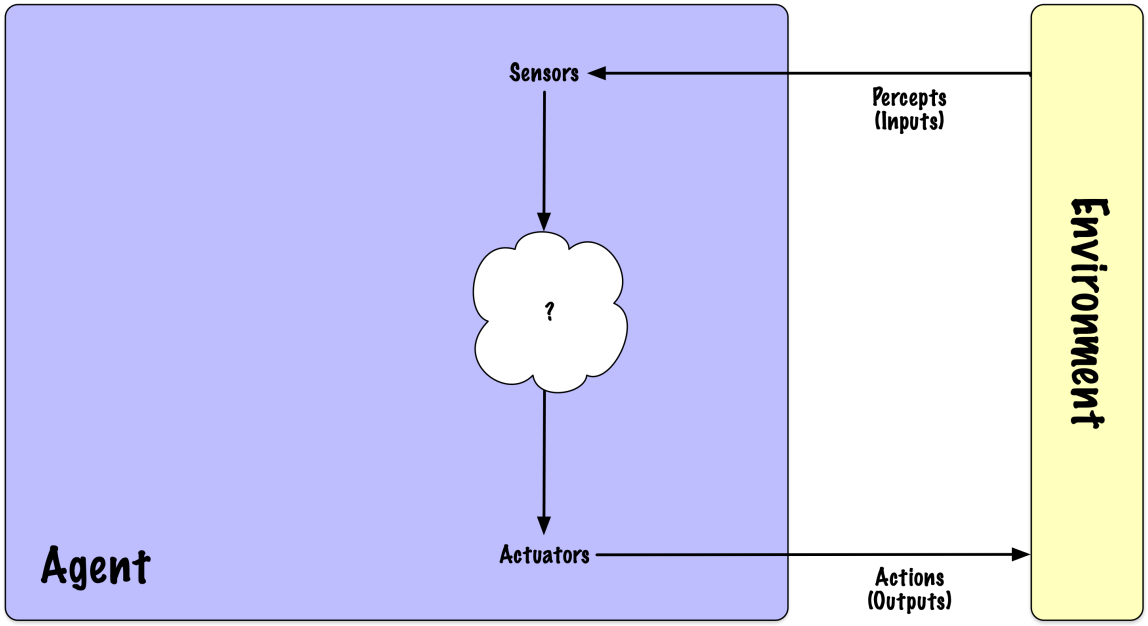
D'autres tâches d'apprentissage non supervisé comprennent :

- la **réduction de dimension**, qui représente les données au moyen d'un nombre réduit de variables tout en préservant une structure utile;
- l'**estimation de densité**, qui modélise les régions où les observations ont tendance à se trouver;
- la **détection d'anomalies**, qui repère les observations très différentes des tendances habituelles.

Ce cours n'approfondira pas ces méthodes. Il suffit ici de reconnaître ce cadre et de le distinguer de l'apprentissage supervisé.

Apprentissage par renforcement

Un **agent** agit dans un **environnement**, observe les conséquences de ses actions et reçoit des **récompenses** numériques.



Objectif : Apprendre une **politique** qui choisit les actions afin de **maximiser la récompense cumulative**.

En **apprentissage par renforcement**, un agent apprend en interagissant de façon répétée avec un environnement. À chaque étape, il reçoit une observation, choisit une action, puis reçoit une récompense numérique ainsi que l'observation suivante.

La récompense évalue les conséquences du comportement, mais elle ne constitue pas une étiquette cible indiquant l'action correcte. Les récompenses peuvent aussi être différées : une action peut influencer les résultats plusieurs étapes plus tard. L'agent doit donc apprendre une **politique** — une stratégie pour choisir les actions — qui donne de bons résultats à long terme, plutôt que de simplement maximiser la récompense immédiate.

Parmi les applications typiques figurent les jeux, la robotique, l'allocation de ressources et les systèmes de commande. Une difficulté centrale consiste à équilibrer l'**exploration** d'actions moins connues et l'**exploitation** de celles qui ont déjà donné de bons résultats.

Ce cours n'approfondira pas les algorithmes d'apprentissage par renforcement. Les étudiants doivent toutefois pouvoir identifier l'agent, l'environnement, les actions, les

observations, les récompenses et l'objectif général.

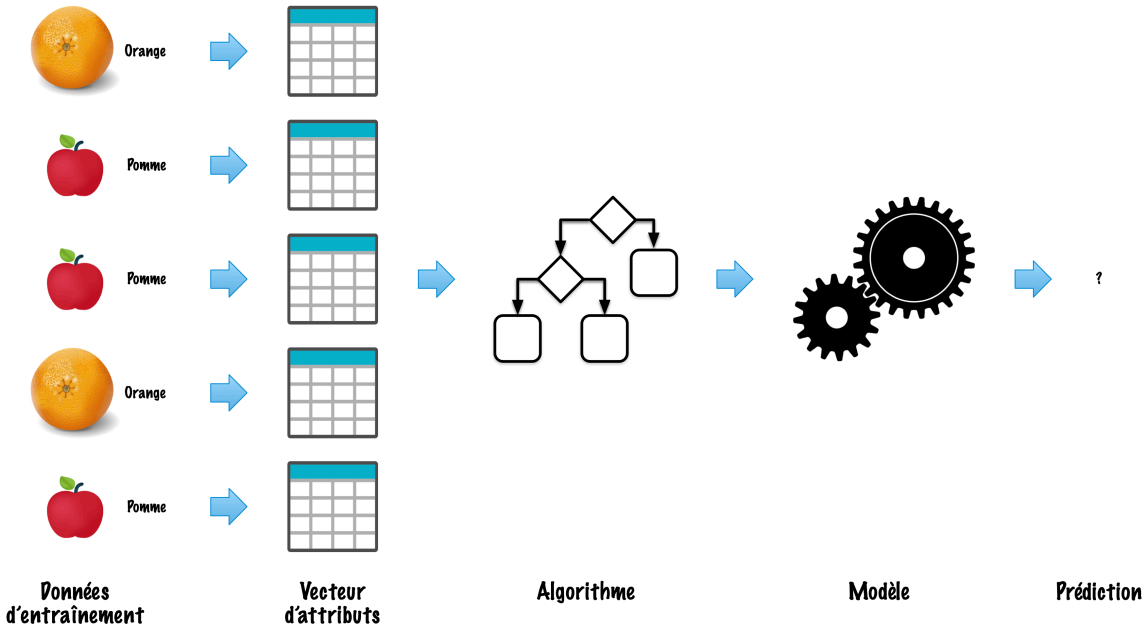
Deux phases

Pour le flux de travail **supervisé** étudié dans ce cours :

1. **Apprentissage** (construction d'un modèle)
2. **Inférence** (utilisation du modèle)

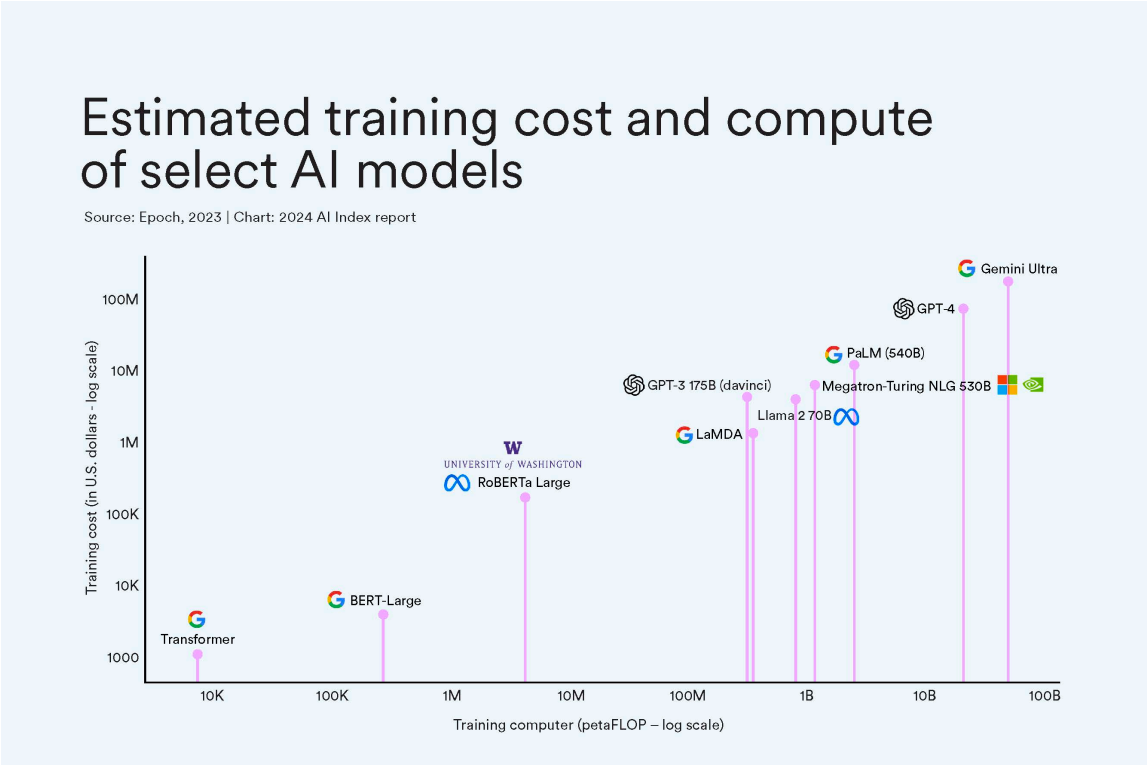
La séparation entre apprentissage et inférence décrit bien le flux de travail supervisé par lots utilisé dans ce cours. Elle n'est pas universelle : en apprentissage par renforcement, par exemple, les actions, la rétroaction et la mise à jour de la politique peuvent être entrelacées.

Apprentissage : construire un modèle



En apprentissage automatique, la phase d'apprentissage est souvent le composant le plus difficile et le plus gourmand en ressources. Cette étape exige une préparation soigneuse des données, ainsi que la sélection et l'entraînement d'un algorithme approprié.

On estime que l'entraînement de certains modèles de pointe contemporains coûte plus de 100 millions de dollars américains.

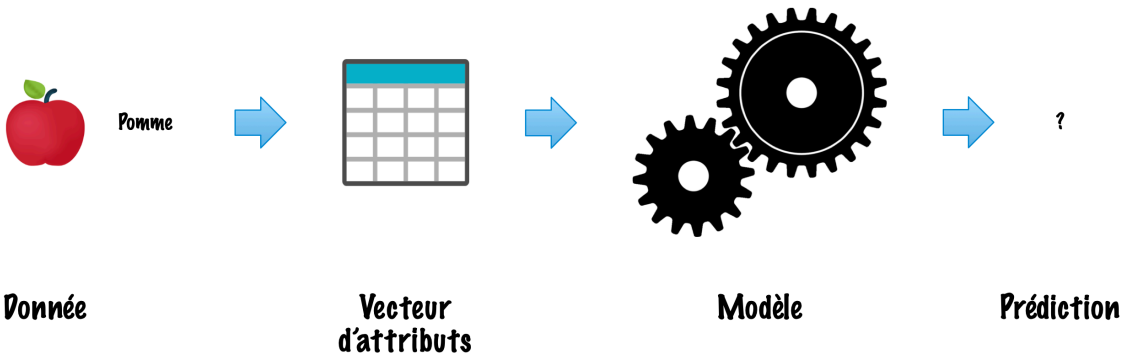


Source : Stanford University Human-Centered Artificial Intelligence 2025 AI Index Report

Le développement de tels modèles nécessite des investissements en infrastructure pouvant atteindre des milliers de milliards de dollars.

- « By the end of 2024, we're aiming to continue to grow our infrastructure build-out that will include **350,000 NVIDIA H100 GPUs** as part of **a portfolio that will feature compute power equivalent to nearly 600,000 H100s**. »
  - Building Meta's GenAI Infrastructure, 2024-03-12.
- OpenAI's Sam Altman Expects to Spend 'Trillions' on Infrastructure, *Bloomberg*, 2025-08-15.

Inférence : utilisation d'un modèle



L'inférence sur un exemple exige généralement moins de ressources que l'entraînement d'un modèle. Son coût total peut néanmoins devenir considérable lorsqu'un modèle répond à de nombreux utilisateurs, et certains systèmes de raisonnement emploient délibérément davantage de calcul au moment de l'inférence.

- OpenAI's o3 Reasoning Models Are Extremely Expensive to Run, Alexandra Tremayne-Pengelly, *Observer*, 2025-04-04. Les estimations médiatiques dépendent fortement de la configuration d'inférence et ne doivent pas être interprétées comme un prix stable par requête.

Carp-e Diem! (exemple)

1. Problème : Vont-ils mordre aujourd'hui ?

Objectif : Développer un modèle prédictif qui classe une journée de pêche comme **médiocre**, **moyenne** ou **excellente**.

Attribution: Gemini 1.5 Flash, 10 septembre, 2024, avec l'invite: "In the style of a children's book, generate the image of a fish with a farmer's hat."

L'apprentissage supervisé consiste à entraîner un modèle sur des données étiquetées afin qu'il puisse **faire des prédictions sur de nouvelles données non vues**.

La tâche spécifique est la **classification**.

"Médiocre", "Moyenne" et "Excellente" sont des **classes** (pour la variable cible).

2. Attributs (caractéristiques)

Les traditions et certains guides de pêche, dont **The Old Farmer's Almanac**, utilisent souvent la **phase de la lune** pour recommander des journées de pêche. Nous la traiterons comme un attribut candidat; son utilité prédictive demeure une question empirique.

- **Phase de la lune (catégorielle)** : 'Nouvelle lune', 'Premier quartier', 'Pleine lune' et 'Dernier quartier'.
- **Prévisions météorologiques (catégorielles)** : 'Pluvieux', 'Nuageux' et 'Ensoleillé'.
- **Température extérieure (numérique)** : La température en Celsius.
- **Température de l'eau (numérique)** : La température de l'eau du lac ou de la rivière.

Évidemment, les applications réelles auront beaucoup plus d'attributs.

3. Données d'entraînement

Exemple	Phase de la lune	Prévisions météorologiques	Température extérieure (°C)	Température de l'eau (°C)	Résultat de la pêche
1	Pleine lune	Ensoleillé	25	22	Excellente
2	Nouvelle lune	Nuageux	18	19	Moyenne
3	Premier quartier	Pluvieux	15	17	Médiocre
4	Dernier quartier	Ensoleillé	30	24	Excellente
5	Pleine lune	Nuageux	20	20	Moyenne
6	Nouvelle lune	Pluvieux	22	21	Médiocre

Il s'agit d'un problème d'**apprentissage supervisé**, puisque la cible est connue pour chaque exemple d'entraînement. Comme les cibles possibles sont des catégories, il s'agit d'une **tâche de classification**.

Ce petit jeu de données est inventé à des fins d'illustration. Il ne contient que six exemples; une étude réelle nécessiterait beaucoup plus d'observations représentatives ainsi qu'une définition rigoureuse du résultat.

Le choix des attributs et la qualité des données sont essentiels pour la performance du modèle. Un attribut tel que la couleur de vos chaussettes n'a probablement pas un grand impact sur les prédictions.

Par conséquent, un projet d'apprentissage automatique commence généralement par une phase exploratoire, qui consiste à analyser les données, examiner les distributions et identifier les corrélations.

3. Données

Phase de la lune	Prévisions météorologiques	Température extérieure (°C)	Température de l'eau (°C)
Pleine lune	Ensoleillé	25	22
Nouvelle lune	Nuageux	18	19
Premier quartier	Pluvieux	15	17
Dernier quartier	Ensoleillé	30	24
Pleine lune	Nuageux	20	20
Nouvelle lune	Pluvieux	22	21

Les données sont souvent présentées sous forme tabulaire (matrice), où chaque ligne représente un **vecteur d'attributs** (*feature vector*), généralement noté x_i, correspondant au i-ème exemple dans l'ensemble d'entraînement.

3. Étiquettes

Résultat de la pêche
Excellente
Moyenne
Médiocre
Excellente

Résultat de la pêche
Moyenne
Médiocre

Les **étiquettes** sont généralement représentées comme un vecteur colonne, avec y_i désignant l'étiquette pour le i-ème exemple.

4. Entraînement du modèle

L'**entraînement du modèle** utilise des exemples étiquetés pour construire un modèle capable d'effectuer des prédictions. Selon l'algorithme, l'entraînement peut **estimer des paramètres**, **construire une structure** ou simplement **mémoriser les exemples**.

4. Entraînement du modèle (suite)

Un modèle simple qui s'ajuste aux six exemples d'entraînement est le suivant :

- Si les prévisions sont **ensoleillées**, prédire **excellente**.
- Si les prévisions sont **nuageuses**, prédire **moyenne**.
- Si les prévisions sont **pluvieuses**, prédire **médiocre**.

Un ajustement parfait sur six exemples ne garantit pas que le modèle prédira correctement de nouvelles journées de pêche.

5. Prédiction

À partir de nouvelles données **inédites**, prédisez la catégorie de la journée de pêche.

. . .

- Phase de la lune : **Nouvelle lune**
- Prévisions météorologiques : **Ensoleillé**
- Température extérieure : **24°C**
- Température de l'eau : **21°C**

. . .

Prédiction : Excellente

Cycle de vie

1. Collecte et préparation des données
2. Ingénierie des attributs

3. Entraînement
4. Évaluation du modèle
5. Déploiement du modèle
6. Surveillance et maintenance

- La **collecte des données** et leur **préparation** sont des processus critiques et laborieux.
- Il doit y avoir une quantité **suffisante** de données.
- Les données doivent être de haute **qualité** ; par exemple, elles ne doivent pas être excessivement bruyantes.
- Il doit y avoir peu de **valeurs manquantes**.
- Plus important encore, les données doivent être **représentatives**. Nous nous attendons à ce que les nouvelles données soient générées par le même processus et aient la même distribution.
- L'importance de cela ne peut être surestimée.
- Pensez à un logiciel de classification d'images qui n'a pas été entraîné sur un échantillon diversifié en termes d'ethnicité, de genre, de taille corporelle ou de statut social.
- Pensez aux applications médicales et aux conséquences de jeux de données qui ne sont pas suffisamment diversifiés.
- L'**ingénierie des attributs** consiste à sélectionner, transformer et créer des variables d'entrée qui rendent les régularités utiles plus faciles à apprendre. Elle peut comprendre la mise à l'échelle des variables numériques, la représentation des variables catégorielles et la création de nouveaux attributs.
- L'apprentissage profond peut apprendre des représentations utiles directement à partir des données, mais la préparation des données et le choix des attributs demeurent importants.
- L'**évaluation du modèle** est le processus d'évaluation des performances d'un modèle d'apprentissage automatique en utilisant des métriques spécifiques, telles que l'exactitude, la précision, le rappel, le F1-score ou l'AUC-ROC. Cela implique généralement de tester le modèle sur un ensemble de validation ou de test séparé pour s'assurer qu'il se généralise bien aux données non vues et qu'il répond aux critères souhaités d'exactitude et de fiabilité.
- Le **déploiement du modèle** consiste à intégrer le modèle entraîné dans une application. De nombreux modèles déployés conservent des paramètres fixes et sont mis à jour périodiquement; d'autres apprennent continuellement à partir de nouvelles données. La stratégie appropriée dépend des coûts, des risques et de la vitesse à laquelle le processus générateur de données évolue.
- La **surveillance et la maintenance** : Les performances du modèle doivent être continuellement surveillées. On observe souvent une **dérive conceptuelle**, nécessitant la réentraînement du système. La détection de spam est un bon exemple. Une fois le système déployé, les spammeurs s'adaptent et trouvent des moyens de contourner les mécanismes de détection de spam mis en place.

Définitions formelles

Apprentissage supervisé (notation)

Le **jeu de données** (« expérience ») est une collection d'exemples étiquetés.

- $\{(x_i, y_i)\}_{i=1}^N$
  - Chaque x_i est un vecteur d'**attributs** (*feature vector*) à D dimensions.
  - $x_i^{(j)}$ est la valeur de l'attribut j pour l'exemple i, où $j \in \{1, \ldots, D\}$ et $i \in \{1, \ldots, N\}$.
  - La **cible** y_i peut être une **classe** appartenant à un ensemble fini $\{1, 2, \ldots, C\}$, un **nombre réel** ou un **objet structuré**, par exemple une séquence, un arbre ou un graphe.

. . .

Problème : À partir du jeu de données, construire un **modèle** capable de prédire la valeur de y pour un x inédit.

Cette notation suit celle du **The Hundred-Page Machine Learning Book** par Andriy Burkov.

Apprentissage supervisé (notation, suite)

- Lorsque la **cible** y_i est une **classe** appartenant à un ensemble fini $\{1, 2, \ldots, C\}$, il s'agit d'une tâche de **classification**.
- Lorsque la **cible** y_i est un **nombre réel**, il s'agit d'une tâche de **régression**.

Dans son livre Why Machines Learn: The Elegant Math Behind Modern AI, Anil Ananthaswamy propose un exemple inspiré de la pandémie de COVID-19.

Dans ce scénario avancé, les hôpitaux du monde entier compilent systématiquement les données des patients depuis les premiers stades de la pandémie de COVID-19. Chaque patient est caractérisé par six variables clés : x_1 représentant l'âge, x_2 indiquant l'indice de masse corporelle, x_3 dénotant la difficulté à respirer (encodée comme 1 pour oui et 0 pour non), x_4 signifiant la présence de fièvre (oui/non), x_5 pour le statut diabétique (oui/non), et x_6 décrivant les résultats du scanner thoracique (0 pour clair, 1 pour infection légère, 2 pour infection sévère). Ces variables forment collectivement un vecteur à six dimensions pour chaque patient.

Les cliniciens ont observé que les patients suivent généralement l'une des deux trajectoires suivantes : soit ils se stabilisent et sont libérés environ trois jours après leur admission, soit leur état se détériore, nécessitant un soutien ventilatoire. Par

conséquent, chaque patient se voit attribuer une variable de résultat, y, où $y = -1$ indique qu'il n'y a pas besoin de soutien ventilatoire après trois jours, et $y = 1$ indique la nécessité de soutien ventilatoire après trois jours.

Pouvez-vous proposer des exemples de tâches de régression? La caractéristique déterminante est que la cible est une quantité numérique. Par exemple :

- estimer le prix de vente d'une maison;
- prévoir la température du lendemain ou la demande d'électricité;
- estimer le temps de trajet ou l'autonomie restante d'une batterie;
- prédire des coûts médicaux;
- estimer le montant que dépensera un client.

Certains problèmes peuvent être formulés de plusieurs façons. Prédire si une personne fera défaut sur un prêt est une tâche de classification; prédire la probabilité de défaut produit plutôt un score numérique.

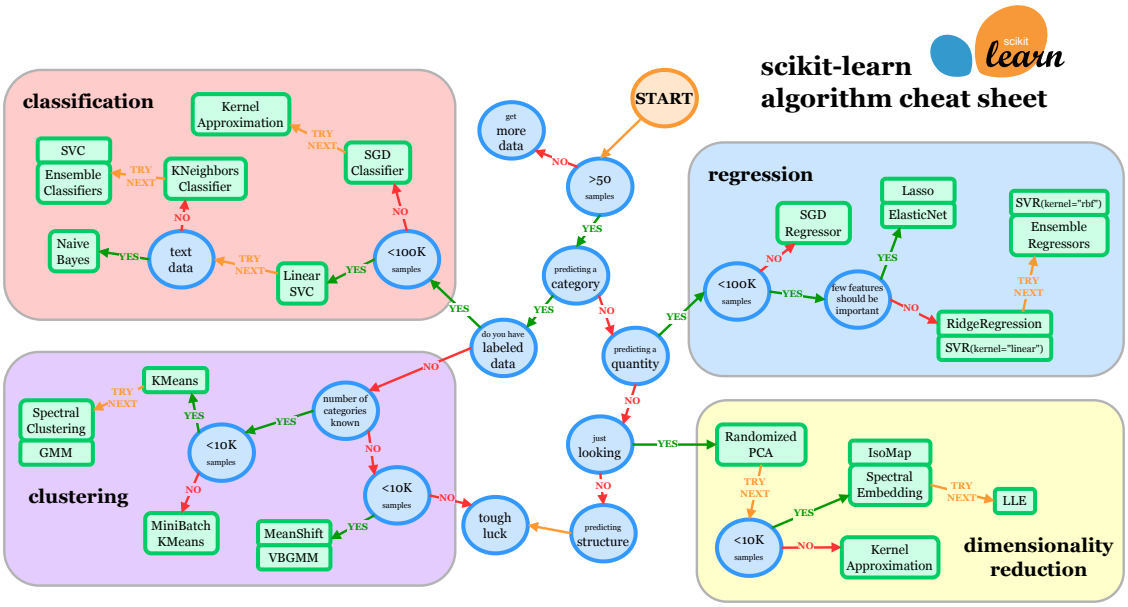
Exemple avec du code

`scikit-learn`

> **scikit-learn.org**
>
> `scikit-learn` est une bibliothèque libre d'apprentissage automatique qui prend en charge l'apprentissage **supervisé** et **non supervisé**. Elle fournit également des outils pour **l'ajustement des modèles**, **le prétraitement des données**, **la sélection de modèles**, **l'évaluation** et de nombreuses autres tâches.
>
> `scikit-learn` fournit **des dizaines d'algorithmes et de modèles intégrés**, appelés estimateurs.
>
> Construite sur NumPy, SciPy et matplotlib.

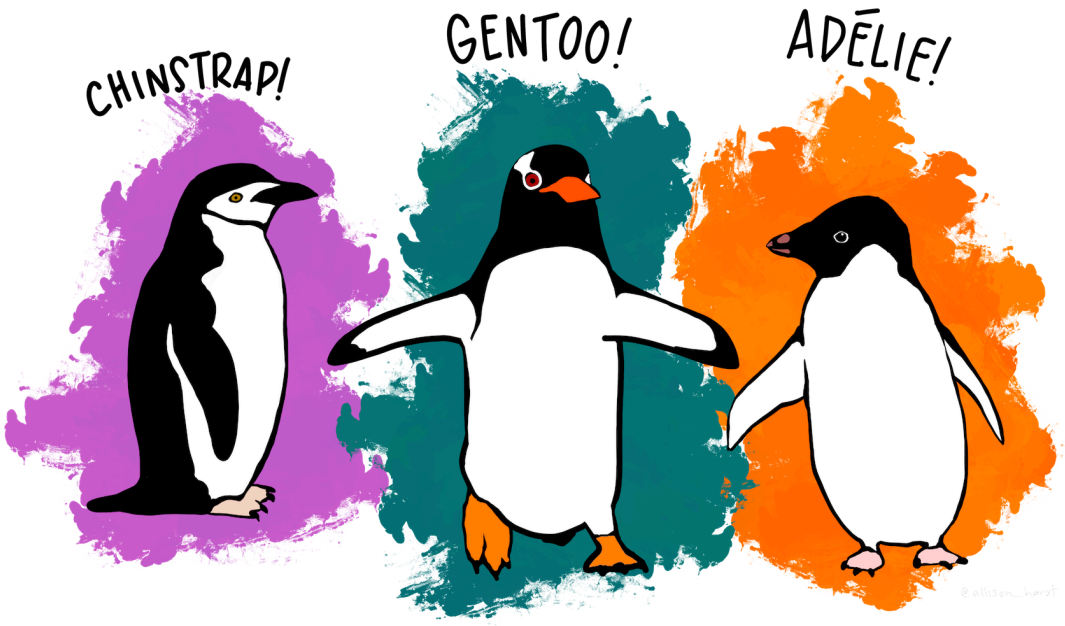
Nous utiliserons `scikit-learn` pour cet exemple et au cours des semaines à venir.

`scikit-learn`



Attribution: Choosing the right estimator

Exemple : manchots de Palmer



Le jeu de données des manchots de Palmer par Allison Horst, Alison Hill, et Kristen Gorman. **Illustration** par @allison_horst

> Le jeu de données des manchots de Palmer par Allison Horst, Alison Hill, et Kristen Gorman a été rendu public pour la première fois sous forme de package R. L'objectif du jeu de données des manchots de Palmer est de remplacer le jeu de données Iris, qui est très surutilisé pour l'exploration et la visualisation des données.

> En utilisant ce package python, vous pouvez facilement charger les manchots de Palmer dans votre environnement python.

Initialement, nous examinerons l'ensemble de l'exemple d'un point de vue général pour fournir une vue d'ensemble claire des étapes impliquées. Par la suite, nous reviendrons sur cet exemple plus en détail.

Exemple : bibliothèque manquante

In [3]:
```
try:
    from palmerpenguins import load_penguins
except ModuleNotFoundError:
    %pip install -q palmerpenguins
    from palmerpenguins import load_penguins
```

Si vous exécutez ce Jupyter Notebook dans Google Colab ou tout autre environnement où la bibliothèque n'est pas préinstallée, procédez à l'installation.

Exemple : chargement des données

In [4]:
```
# Il est d'usage d'utiliser X et y pour les données et les étiquettes

X, y = load_penguins(return_X_y = True)

# Retirer les lignes qui contiennent des mesures manquantes

X = X.dropna()
y = y.loc[X.index]
```

Deux manchots ont des mesures manquantes. Pour ce premier exemple, nous retirons ces lignes avant l'entraînement.

Exemple : utilisation d'un arbre de décision

In [5]:
```
from sklearn import tree

clf = tree.DecisionTreeClassifier(criterion="entropy", random_state=42)
```

Parmi les classificateurs disponibles figurent les **arbres de décision**, les **machines à vecteurs de support**, les **k plus proches voisins**, la **régression logistique** et les **réseaux de neurones**.

Exemple : entraînement

In [6]:
```
# Entraînement
```

```
clf = clf.fit(X, y)
```

Les estimateurs de scikit-learn partagent une interface cohérente. Les classificateurs offrent notamment les méthodes `fit`, `predict` et `score`.

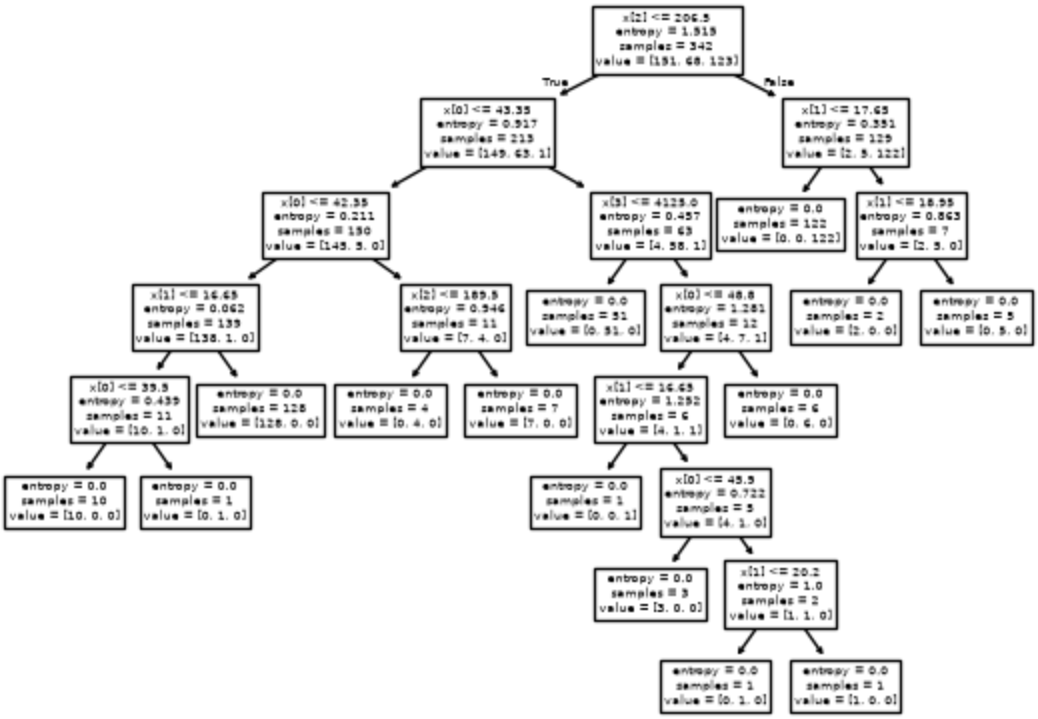
Le `DecisionTreeClassifier` de scikit-learn construit un arbre en divisant récursivement le jeu de données. Avec `criterion="entropy"`, il sélectionne les seuils qui produisent la plus grande réduction de l'entropie pondérée.

1. **Initialisation** : L'algorithme commence avec l'ensemble du jeu de données comme nœud racine.
2. **Critère de division** : À chaque nœud, l'algorithme évalue des seuils candidats pour les attributs numériques et retient la division qui sépare le mieux les classes.
3. **Division Récursive** : Le jeu de données est divisé en sous-ensembles basés sur l'attribut et le seuil sélectionnés, créant des nœuds enfants. Ce processus est répété de manière récursive pour chaque nœud enfant.
4. **Conditions d'arrêt** : La division s'arrête lorsqu'un nœud est pur, qu'aucune division candidate n'améliore l'objectif, ou qu'une limite de profondeur ou de taille s'applique.
5. **Nœuds Terminaux** : Une fois la division terminée, chaque nœud terminal se voit attribuer une étiquette de classe basée sur la classe majoritaire des échantillons dans ce nœud.

L'arbre résultant peut ensuite être utilisé pour classifier de nouveaux échantillons en parcourant de la racine à un nœud terminal, en suivant les règles de décision définies à chaque nœud.

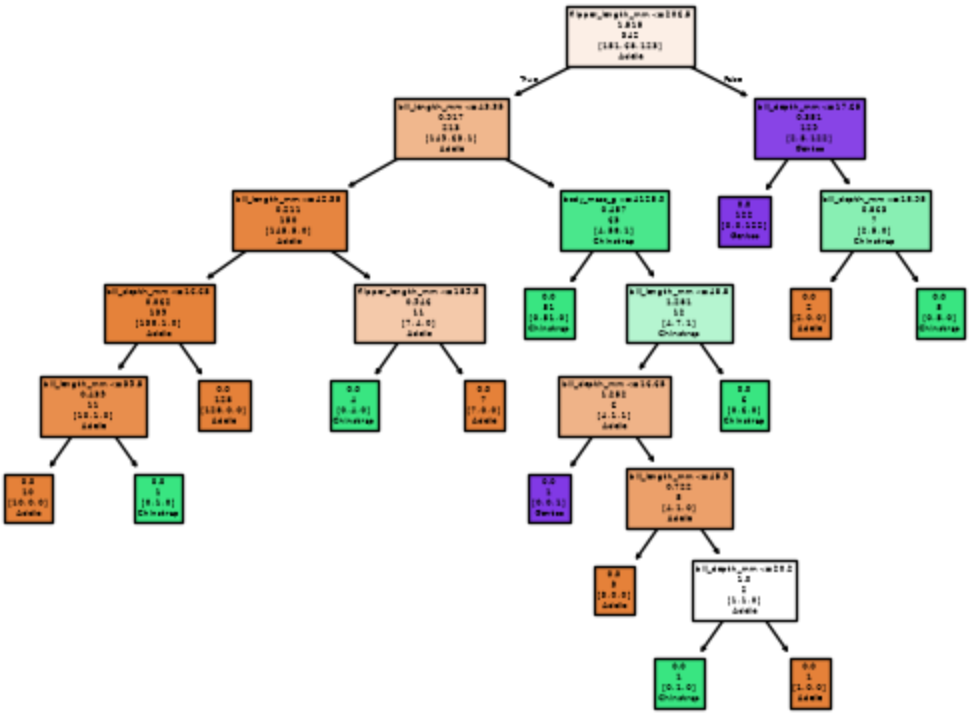
Exemple : visualisation (1/2)

In [7]:
```
import matplotlib.pyplot as plt

tree.plot_tree(clf)
plt.show()
```



Exemple : visualisation (2/2)

In [8]:
```
target_names = clf.classes_

tree.plot_tree(clf,
               feature_names = X.columns,
               class_names = target_names,
               label = 'none',
               filled = True)
plt.show()
```



Exemple : prédiction

In [9]:
```
import pandas as pd

# Créer deux nouveaux exemples

column_names = ['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'boc
new_penguins = pd.DataFrame(
    [[34.2, 17.9, 186.8, 2945.0], [51.0, 15.2, 223.7, 5560.0]],
    columns=column_names,
)

# Prédiction

predictions = clf.predict(new_penguins)

# Affichage des étiquettes prédites pour nos deux exemples

print(predictions)
```

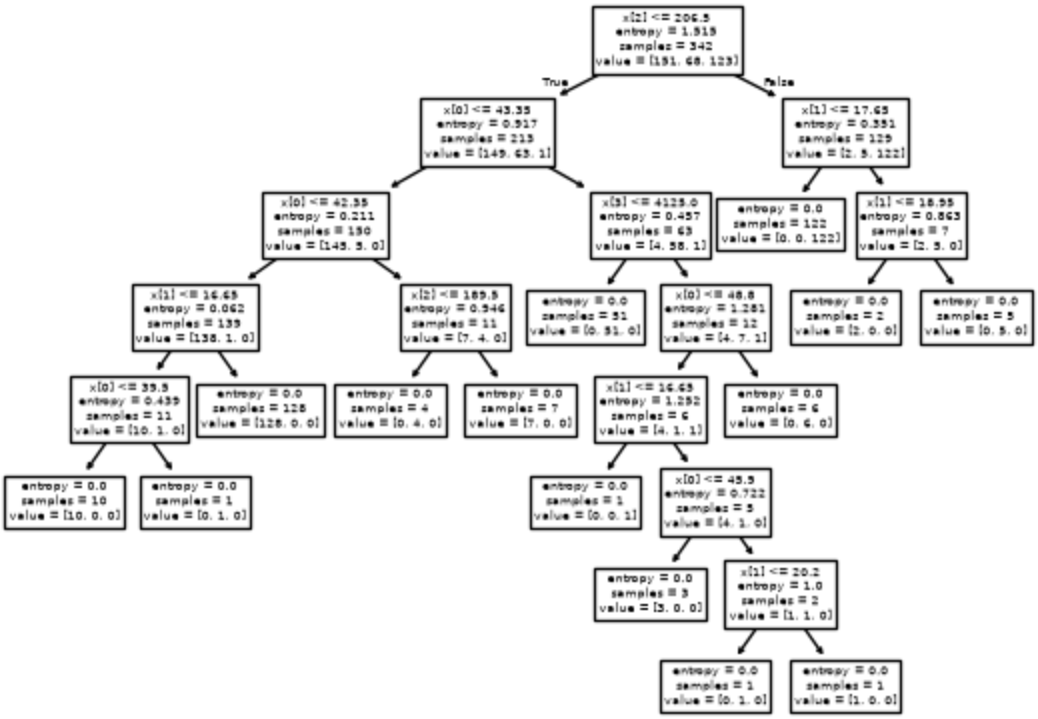
```
['Adelie' 'Gentoo']
```

Exemple : exemple complet

In [10]:
```
X, y = load_penguins(return_X_y = True)
X = X.dropna()
y = y.loc[X.index]
clf = tree.DecisionTreeClassifier(criterion="entropy", random_state=42)
clf = clf.fit(X, y)
tree.plot_tree(clf)
```

```
new_penguins = pd.DataFrame(
    [[34.2, 17.9, 186.8, 2945.0], [51.0, 15.2, 223.7, 5560.0]],
    columns=column_names,
)
print(clf.predict(new_penguins))
```

```
['Adelie' 'Gentoo']
```



Exemple : performance apparente

In [11]:
```
from sklearn.metrics import classification_report, accuracy_score

# Faire des prédictions

y_pred = clf.predict(X)

# Évaluer le modèle

accuracy = accuracy_score(y, y_pred)
report = classification_report(y, y_pred, labels=clf.classes_, target_names=

print(f'Exactitude : {accuracy:.2f}')
print('Rapport de classification :')
print(report)
```

```
Exactitude : 1.00
Rapport de classification :
              precision    recall  f1-score   support

      Adelie       1.00      1.00      1.00       151
   Chinstrap       1.00      1.00      1.00        68
      Gentoo       1.00      1.00      1.00       123

    accuracy                           1.00       342
   macro avg       1.00      1.00      1.00       342
weighted avg       1.00      1.00      1.00       342
```

Exemple : discussion

Nous avons démontré un exemple complet :

- Chargement des données
- Sélection d'un classificateur
- Entraînement du modèle
- Visualisation du modèle
- Faire une prédiction

Cependant, plusieurs simplifications ont été faites tout au long de ce processus.

Exemple : attendez une minute!

In [12]:
```python
from sklearn.metrics import classification_report, accuracy_score

# Faire des prédictions

y_pred = clf.predict(X)

# Évaluer le modèle

accuracy = accuracy_score(y, y_pred)
report = classification_report(y, y_pred, labels=clf.classes_, target_names=

print(f'Exactitude : {accuracy:.2f}')
print('Rapport de classification :')
print(report)
```

> **Important**
>
> Cet exemple est trompeur, voire erroné !

La performance de ce classificateur semble parfaite à première vue. Cependant, l'est-elle vraiment ?

Ces prédictions portent sur les exemples qui ont servi à l'entraînement. Il s'agit de l'**exactitude d'entraînement**, et non d'une estimation de la performance sur des données inédites.

Exemple : exploration

In [13]:
```
penguins = load_penguins()
```

...

In [14]:
```
type(penguins)
```

```
pandas.DataFrame
```

...

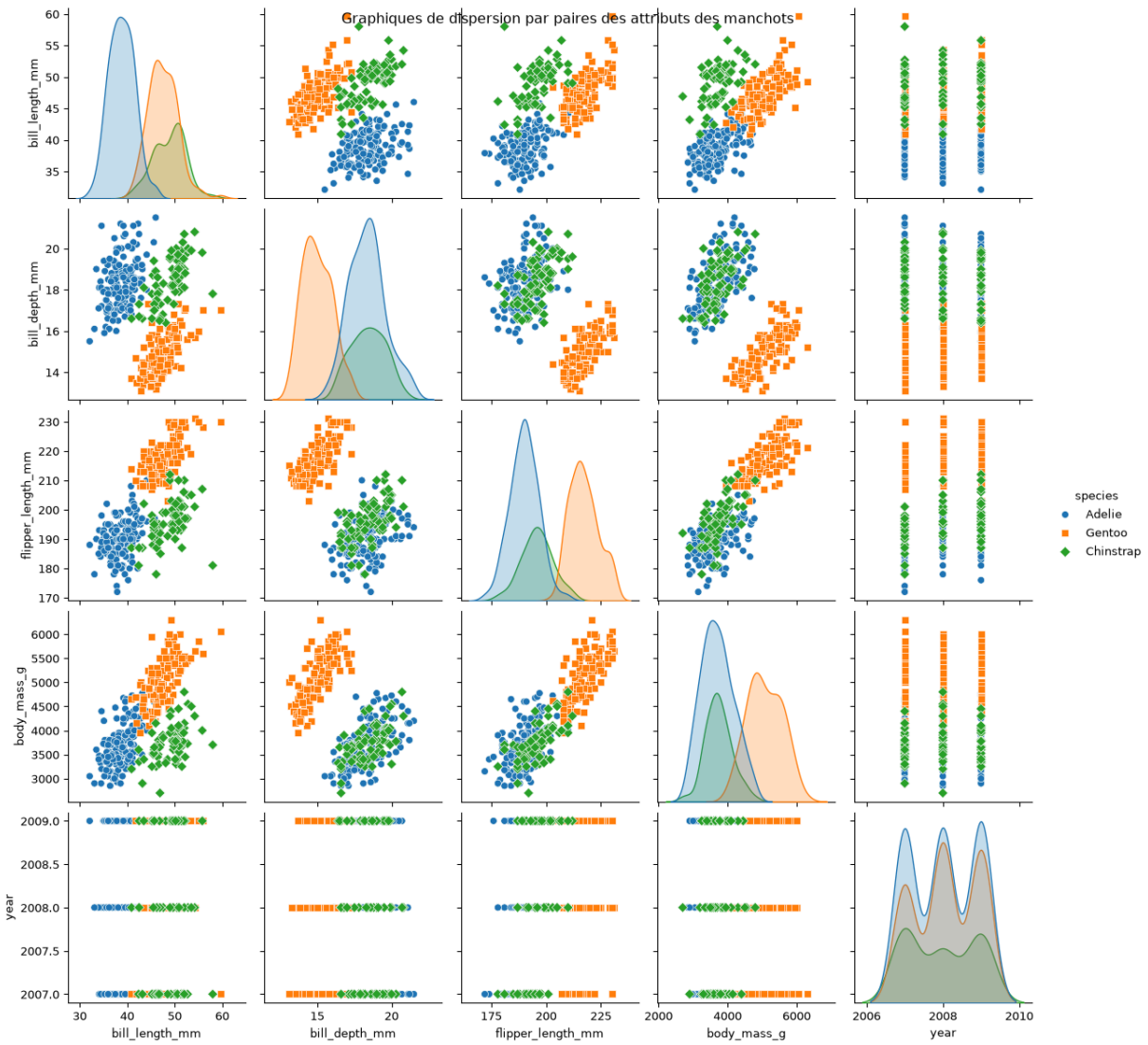
In [15]:
```
penguins.head()
```

Exemple : exploration

In [16]:
```
penguins.describe()
```

Exemple : utilisation de `seaborn`

In [17]:
```
import seaborn as sns

# Graphiques par paires avec seaborn

sns.pairplot(penguins, hue='species', markers=["o", "s", "D"])
plt.suptitle("Graphiques de dispersion par paires des attributs des manchots
plt.show()
```



Quelles informations peuvent être tirées de l'examen des graphiques ?

La figure présente les graphiques par paires des attributs des manchots de Palmer; la diagonale montre la distribution de chaque attribut. Chaque point représente un exemple (x), et sa couleur indique son étiquette (y).

Considérons d'abord les éléments diagonaux :

- Est-il possible de classer les exemples en utilisant un seul attribut ?
- Nous observons que chaque attribut seul ne peut pas distinguer entre les classes.
- Cependant, les graphiques qui font intervenir la **profondeur du bec**, la **masse corporelle** ou la **longueur de la nageoire** permettent de distinguer les **Gentoo** des deux autres espèces, même si les **Adélie** et les **Chinstrap** se chevauchent encore.

La classe **Gentoo** forme fréquemment un groupe distinct.

Exemple : jeux d'entraînement et de test

In [18]:
```
from sklearn.model_selection import train_test_split

# Diviser le jeu de données en ensembles d'entraînement et de test

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=7,
    stratify=y,
)
```

Notre premier arbre de décision a été construit et évalué sur l'ensemble du jeu de données. Il peut très bien s'ajuster à ces exemples, mais cela ne nous indique pas avec quelle exactitude il prédira des **données inédites**.

Nous aurons beaucoup plus à dire sur les tests dans les semaines à venir.

Pour simplifier le récit, les diapositives ont exploré l'ensemble du jeu de données avant d'introduire la séparation. Dans une expérience rigoureuse, le jeu de test doit être mis de côté avant de faire des choix de modélisation guidés par les données, puis demeurer intact jusqu'à l'évaluation finale.

`random_state` rend la partition pseudoaléatoire reproductible. En changeant sa valeur, on change les exemples réservés au test et, possiblement, l'exactitude mesurée. `stratify=y` conserve approximativement les proportions des classes dans les deux sous-ensembles.

Pratique : Essayez différentes valeurs de `random_state`. Qu'observez-vous, et pourquoi?

Exemple : création d'un nouveau classificateur

In [19]:
```
clf = tree.DecisionTreeClassifier(criterion="entropy", random_state=42)
```

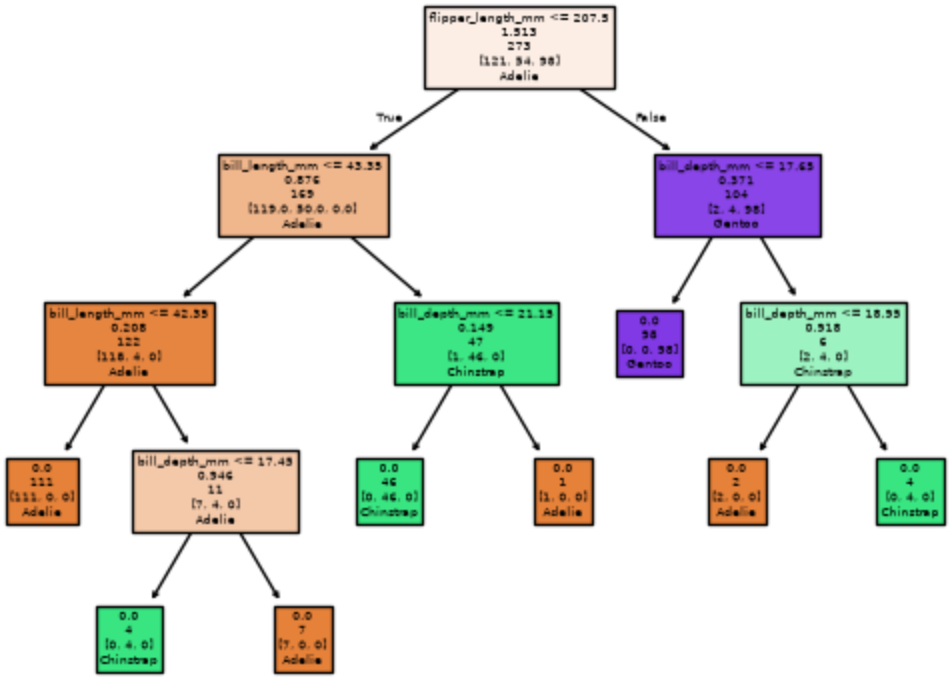
Exemple : entraînement du nouveau classificateur

In [20]:
```
clf.fit(X_train, y_train)
```

Exemple : visualisation de l'arbre

In [21]:
```
tree.plot_tree(clf,
               feature_names = X.columns,
               class_names = target_names,
               label = 'none',
```

```
                filled = True)
plt.show()
```



Exemple : prédictions

In [22]:
```
# Faire des prédictions
y_pred = clf.predict(X_test)
```

Exemple : mesure de la performance

In [23]:
```
# Évaluer le modèle
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(
    y_test,
    y_pred,
    labels=clf.classes_,
    target_names=clf.classes_,
)

print(f'Exactitude : {accuracy:.2f}')
print('Rapport de classification :')
print(report)
```

```
Exactitude : 0.91
Rapport de classification :
              precision    recall  f1-score   support

      Adelie       0.96      0.90      0.93        30
   Chinstrap       0.72      0.93      0.81        14
      Gentoo       1.00      0.92      0.96        25

    accuracy                           0.91        69
   macro avg       0.90      0.92      0.90        69
weighted avg       0.93      0.91      0.92        69
```

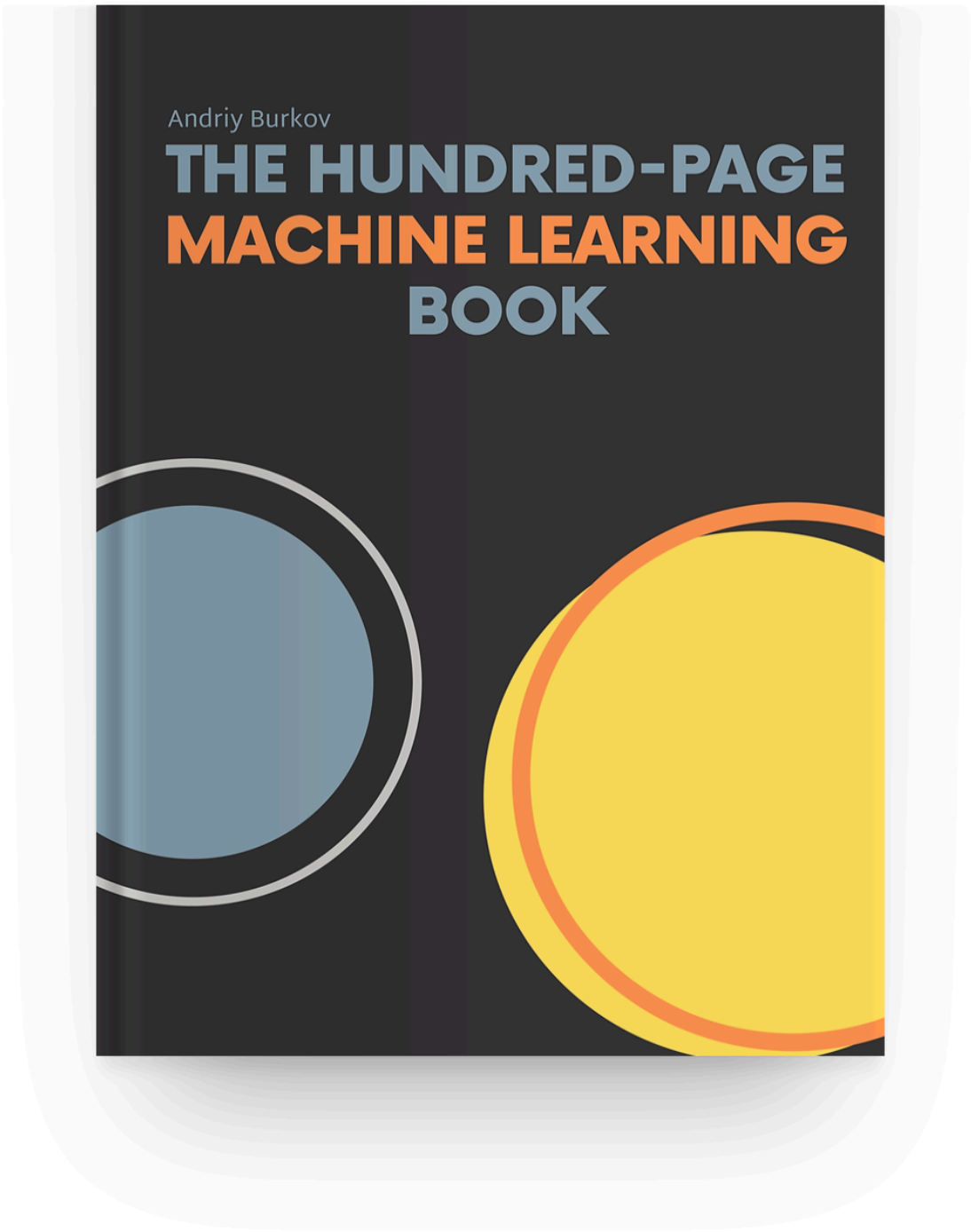
Voici une discussion sur la persistance des modèles pour les modèles scikit-learn.

Résumé

- Nous avons distingué l'apprentissage **supervisé**, **non supervisé** et **par renforcement** selon la rétroaction disponible.
- Nous avons identifié les exemples, les attributs et les cibles d'un problème supervisé.
- Nous avons appliqué le flux de travail de scikit-learn au jeu de données des manchots de Palmer.
- Nous avons exploré les données, entraîné un classificateur et effectué des prédictions.
- Nous avons utilisé un jeu de test distinct pour estimer la performance sur des données inédites.

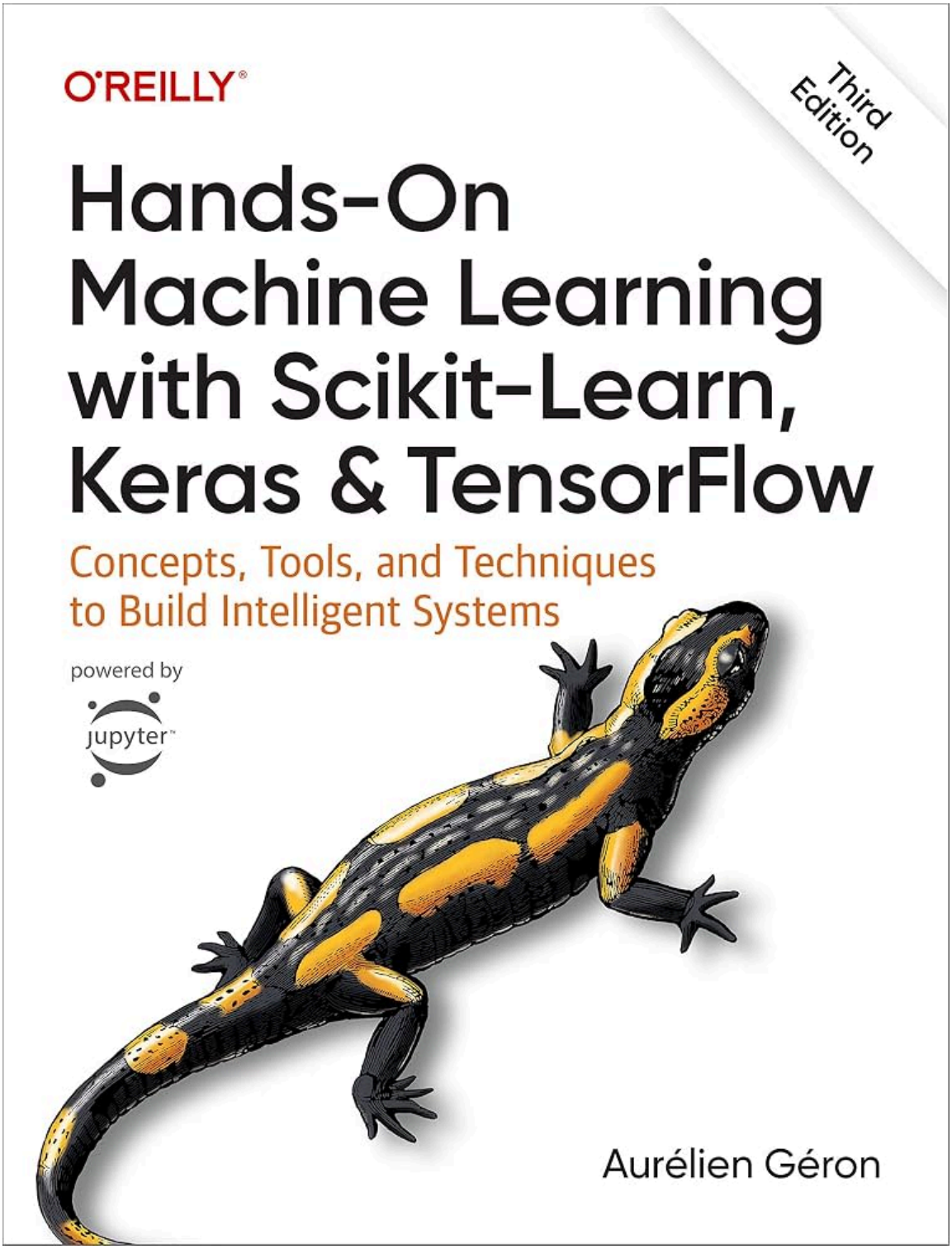
Prologue

Lectures supplémentaires (1/3)



- **The Hundred-Page Machine Learning Book** (Burkov 2019) est un manuel succinct et ciblé qui peut être lu en une semaine, ce qui en fait une excellente ressource d'introduction.
- Disponible sous un modèle « lire d'abord, acheter ensuite », permettant aux lecteurs d'évaluer son contenu avant de l'acheter.
- Son auteur, Andriy Burkov, a obtenu son doctorat en IA à l'Université Laval.

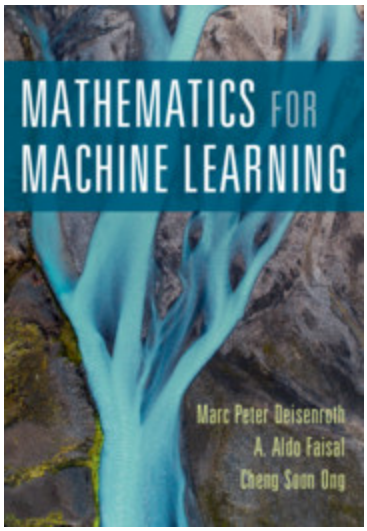
Lectures supplémentaires (2/3)



- **Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow** (Géron 2022) fournit des exemples pratiques et exploite des frameworks Python prêts pour la production.
  - La couverture complète inclut non seulement les modèles, mais aussi des bibliothèques pour le réglage des hyperparamètres, le prétraitement des données et la visualisation.
  - Exemples de code et solutions aux exercices disponibles sous forme de Jupyter Notebooks sur GitHub.

- Aurélien Géron est un ancien chef de produit YouTube, qui a dirigé la classification vidéo pour la recherche et la découverte.

Lectures supplémentaires (3/3)



- **Mathematics for Machine Learning** (Deisenroth et al. 2020) vise à fournir les compétences mathématiques nécessaires pour lire des livres sur l'apprentissage automatique.
- PDF du livre
- "Ce livre offre une excellente couverture de tous les concepts mathématiques de base pour l'apprentissage automatique. J'ai hâte de le partager avec les étudiants, les collègues et toute personne intéressée à acquérir une compréhension solide des fondamentaux." Joelle Pineau, Université McGill et Facebook

Références

Burkov, Andriy. 2019. *The Hundred-Page Machine Learning Book*. Andriy Burkov.

Deisenroth, Marc Peter, A. Aldo Faisal, et Cheng Soon Ong. 2020. *Mathematics for Machine Learning*. Cambridge University Press. https://doi.org/10.1017/9781108679930.

Géron, Aurélien. 2022. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 3e éd. O'Reilly Media, Inc.

Kingsford, C, et Steven L Salzberg. 2008. « What are decision trees? » *Nature biotechnology* 26 (9): 1011-13. https://doi.org/10.1038/nbt0908-1011.

Mitchell, Tom M. 1997. *Machine Learning*. McGraw-Hill.

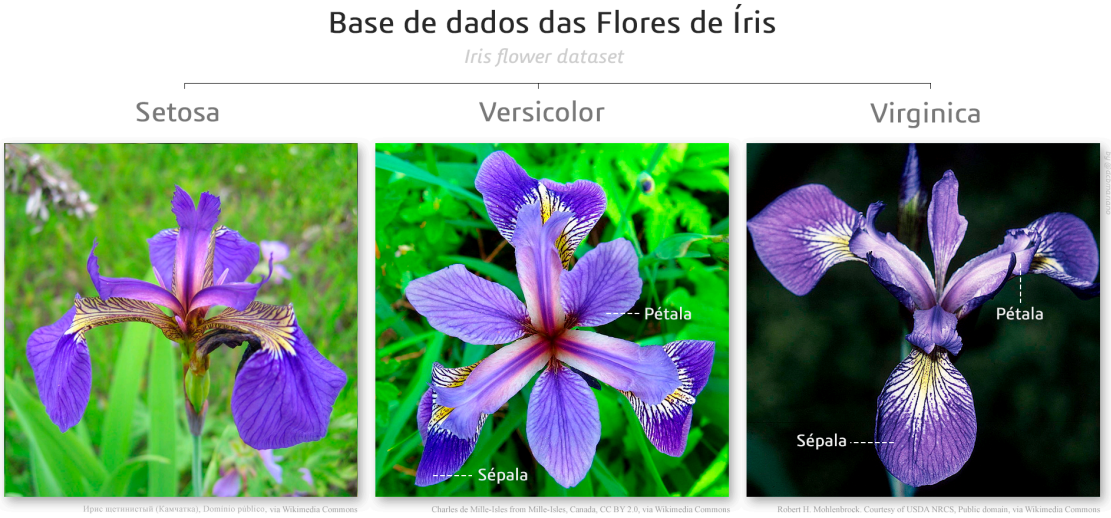
Russell, Stuart, et Peter Norvig. 2020. *Artificial Intelligence: A Modern Approach*. 4e éd. Pearson. http://aima.cs.berkeley.edu/.

Prochain cours

- Arbres de décision et entropie
- Frontières de décision et régression logistique
- K plus proches voisins

Annexe : jeu de données Iris

Exemple : jeu de données Iris



Attribution: Diego Mariano, CC BY-SA 4.0, via Wikimedia Commons. Voir ici pour des informations sur cet ensemble de données.

Exemple : chargement des données

In [24]:
```
from sklearn.datasets import load_iris

# Charger le jeu de données Iris

iris = load_iris()
```

De manière pratique, scikit-learn est fourni avec des ensembles de données jouets et réels pour faciliter l'expérimentation. Voir ici pour une description de **load_iris**.

Lorsque vous utilisez votre propre jeu de données, vous devez généralement le charger à partir d'un fichier ou d'une source en ligne, comme dans le cahier sur la température de la rivière des Outaouais.

Exemple : utilisation d'un arbre de décision

In [25]:
```
from sklearn import tree

clf = tree.DecisionTreeClassifier(criterion="entropy", random_state=42)
```

Parmi les classificateurs disponibles figurent les **arbres de décision**, les **machines à vecteurs de support**, les **k plus proches voisins**, la **régression logistique** et les **réseaux de neurones**.

Dans un `DecisionTreeClassifier`, chaque nœud interne de l'arbre représente une décision basée sur un attribut, chaque branche représente le résultat de cette décision, et chaque nœud terminal représente une étiquette de classe. L'arbre de décision fait des prédictions en parcourant de la racine à un nœud terminal, en suivant les règles de décision définies à chaque nœud. Les arbres de décision sont intuitifs et faciles à interpréter mais peuvent être sujets au surapprentissage s'ils ne sont pas correctement régulés.

Exemple : entraînement

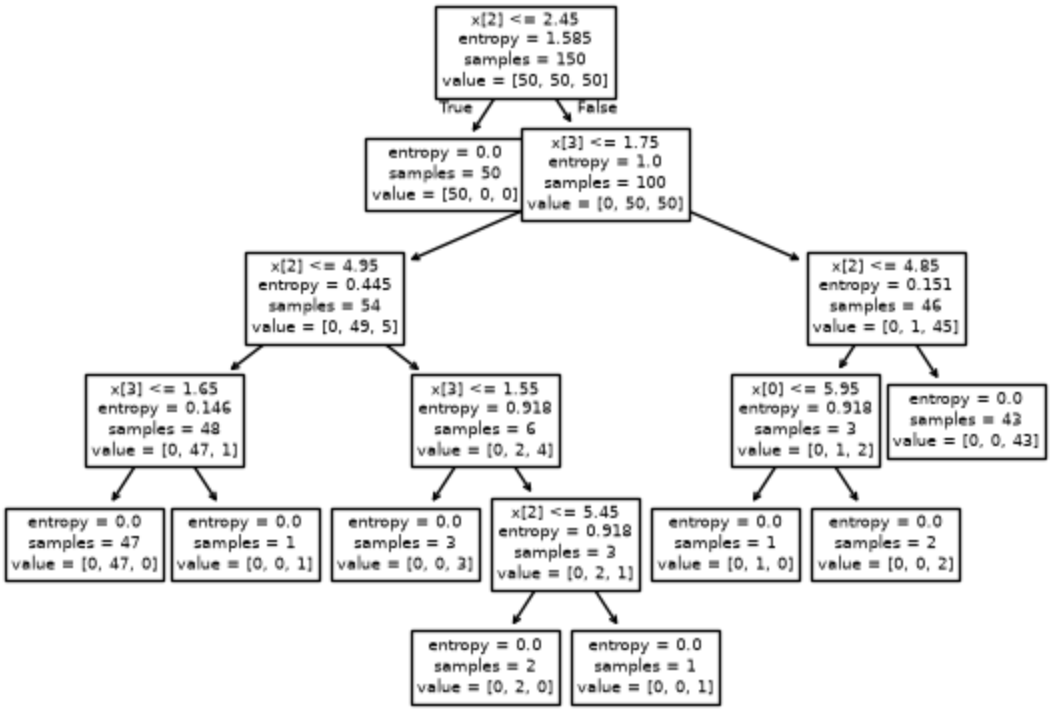
In [26]:
```
# Par convention, X désigne les données et y les cibles

X, y = iris.data, iris.target

# Entraînement

clf = clf.fit(X, y)
```

Les estimateurs de scikit-learn partagent une interface cohérente. Les classificateurs offrent notamment les méthodes `fit`, `predict` et `score`.

Avec `criterion="entropy"`, `DecisionTreeClassifier` construit un arbre en choisissant récursivement les seuils qui réduisent l'entropie pondérée. Chaque division crée deux nœuds enfants. Le processus se poursuit jusqu'à ce qu'aucune division utile ne subsiste ou qu'une autre condition d'arrêt soit satisfaite. Chaque feuille prédit la classe majoritaire parmi les exemples d'entraînement qui l'atteignent.

Exemple : visualisation de l'arbre (1/2)

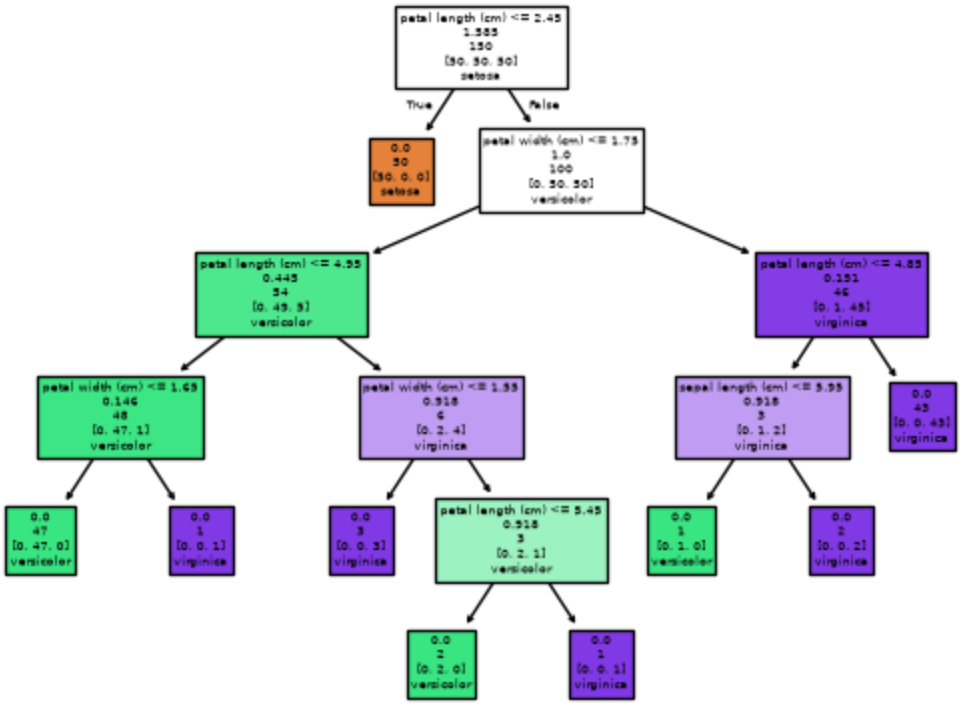
In [27]:
```
import matplotlib.pyplot as plt

tree.plot_tree(clf)
plt.show()
```



Exemple : visualisation de l'arbre (2/2)

In [28]:

```
tree.plot_tree(clf,
               feature_names=iris.feature_names,
               class_names=iris.target_names,
               label='none',
               filled=True)
plt.show()
```



Dans un `DecisionTreeClassifier`, chaque nœud interne de l'arbre représente une décision basée sur un attribut, chaque branche représente le résultat de cette décision, et chaque nœud feuille représente une étiquette de classe. L'arbre de décision fait des prédictions en parcourant de la racine à un nœud feuille, suivant les règles de décision définies à chaque nœud. Les arbres de décision sont intuitifs et faciles à interpréter, mais peuvent être sujets au surapprentissage s'ils ne sont pas correctement régulés.

Pour construire un arbre de décision, la méthode `fit` suit essentiellement les étapes suivantes :

1. **Évaluer les divisions candidates :** Pour chaque attribut numérique, considérer des seuils entre les valeurs observées.
2. **Sélectionner la meilleure division :** Retenir l'attribut et le seuil qui produisent la plus faible entropie pondérée.
3. **Diviser le jeu de données :** Envoyer chaque exemple vers l'enfant gauche ou droit selon le seuil choisi.
4. **Répéter récursivement :** Appliquer le même processus dans chaque enfant. Un attribut numérique peut être réutilisé plus bas dans l'arbre.
5. **Appliquer les conditions d'arrêt :** Arrêter lorsqu'un nœud est pur, qu'aucune division candidate n'améliore l'objectif, ou qu'une limite de profondeur ou de taille est atteinte.
6. **Attribuer les prédictions :** Chaque feuille prédit la classe majoritaire des exemples qu'elle contient.

Ce processus aboutit à un arbre où chaque chemin de la racine à une feuille représente une règle de classification.

Dans la figure ci-dessus, les nœuds de décision contiennent les informations suivantes :

- La règle de décision, par exemple `largeur des pétales (cm) <= 0.8`
- L'entropie du nœud.
- Le nombre d'exemples dans le sous-ensemble correspondant à ce nœud de l'arbre.
- Le nombre d'exemples pour chacune des classes, dans le sous-ensemble correspondant à ce nœud de l'arbre.
- Une prédiction.

Les arbres de décision ajoutent progressivement des nœuds guidés par les exemples d'entraînement étiquetés. Une division utile n'a pas besoin de séparer parfaitement toutes les classes; elle doit réduire l'entropie pondérée des enfants. Dans ce jeu de données, la règle `largeur des pétales (cm) <= 0.8` sépare Setosa des deux autres espèces. Lorsque la condition est vraie, tous les exemples de l'enfant gauche sont des Setosa. L'enfant droit contient des Versicolor et des Virginica, mais aucun Setosa.

Voir :

- Kingsford et Salzberg (2008), vous pouvez accéder à l'article ici, en HTML ou PDF, à partir d'un ordinateur avec une adresse IP de l'uOttawa.

Exemple : prédiction

In [29]:
```
# Créer deux nouveaux exemples
# 'sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width

new_irises = [[5.1, 3.5, 1.4, 0.2], [6.7, 3.0, 5.2, 2.3]]

# Prédiction

predictions = clf.predict(new_irises)

# Afficher les classes prédites

print(iris.target_names[predictions])
```

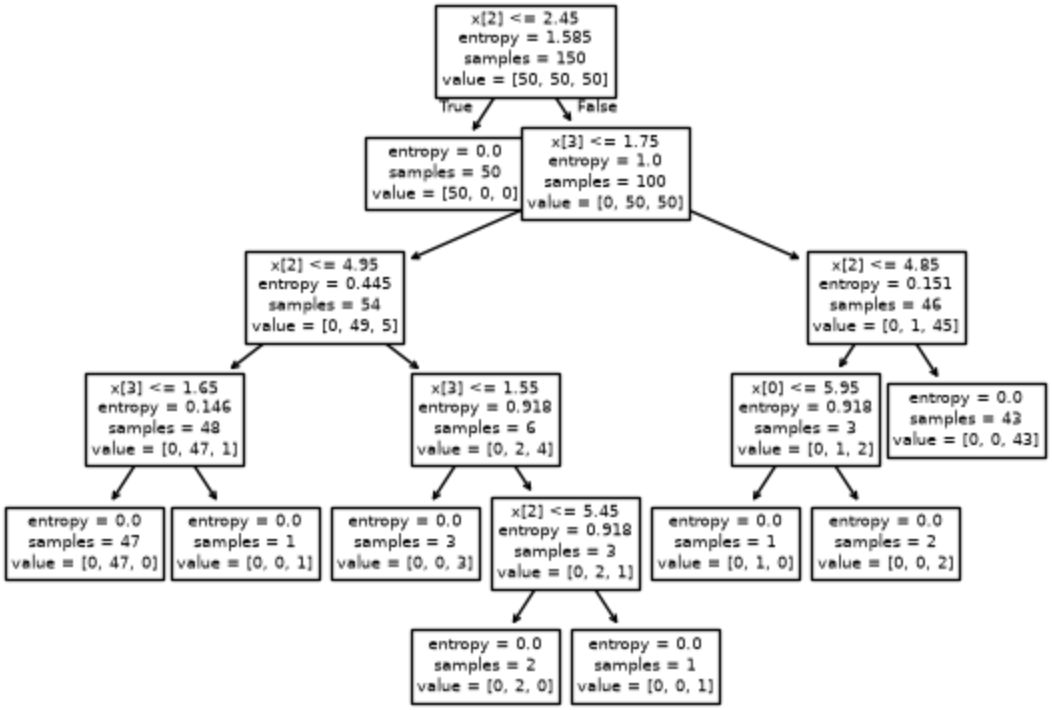
```
['setosa' 'virginica']
```

Exemple : complet

In [30]:
```
iris = load_iris()
clf = tree.DecisionTreeClassifier(criterion="entropy", random_state=42)
X, y = iris.data, iris.target
clf = clf.fit(X, y)
tree.plot_tree(clf)
new_irises = [[5.1, 3.5, 1.4, 0.2], [6.7, 3.0, 5.2, 2.3]]
```

```
predictions = clf.predict(new_irises)
print(iris.target_names[predictions])
```

```
['setosa' 'virginica']
```



Exemple : performance

In [31]:
```
from sklearn.metrics import classification_report, accuracy_score

# Faire des prédictions

y_pred = clf.predict(X)

# Évaluer le modèle

accuracy = accuracy_score(y, y_pred)
report = classification_report(y, y_pred, target_names=iris.target_names)

print(f'Exactitude : {accuracy:.2f}')
print('Rapport de classification :')
print(report)
```

```
Exactitude : 1.00
Rapport de classification :
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        50
  versicolor       1.00      1.00      1.00        50
   virginica       1.00      1.00      1.00        50

    accuracy                           1.00       150
   macro avg       1.00      1.00      1.00       150
weighted avg       1.00      1.00      1.00       150
```

La performance de ce classificateur semble parfaite à première vue. Mais l'est-elle vraiment?

Exemple : discussion

Nous avons vu un exemple complet :

- Chargement des données
- Sélection d'un classificateur
- Entraînement du modèle
- Visualisation du modèle
- Réalisation d'une prédiction

Cependant, plusieurs simplifications ont été faites au cours de ce processus.

Exemple : deuxième tentative

In [32]:
```python
from sklearn.metrics import classification_report, accuracy_score

# Faire des prédictions

y_pred = clf.predict(X)

# Évaluer le modèle

accuracy = accuracy_score(y, y_pred)
report = classification_report(y, y_pred, target_names=iris.target_names)

print(f'Exactitude : {accuracy:.2f}')
print('Rapport de classification :')
print(report)
```

> **Important**
>
> Cet exemple est trompeur, voire erroné!

La performance de ce classificateur semble parfaite à première vue. Mais l'est-elle vraiment?

Exemple : exploration

In [33]:
```
print(f'Description du jeu de données :\n{iris["DESCR"]}\n')
```

Description du jeu de données :
.. _iris_dataset:

Iris plants dataset

Data Set Characteristics:

:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive attributes and the class
:Attribute Information:
    - sepal length in cm
    - sepal width in cm
    - petal length in cm
    - petal width in cm
    - class:
            - Iris-Setosa
            - Iris-Versicolour
            - Iris-Virginica

:Summary Statistics:

============== ==== ==== ======= ===== ====================
                Min Max   Mean   SD   Class Correlation
============== ==== ==== ======= ===== ====================
sepal length: 4.3 7.9   5.84   0.83   0.7826
sepal width: 2.0 4.4   3.05   0.43   -0.4194
petal length: 1.0 6.9   3.76   1.76   0.9490 (high!)
petal width: 0.1 2.5   1.20   0.76   0.9565 (high!)
============== ==== ==== ======= ===== ====================

:Missing Attribute Values: None
:Class Distribution: 33.3% for each of 3 classes.
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is take
n
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature. Fisher's paper is a classic in the field an
d
is referenced frequently to this day. (See Duda & Hart, for example.) The
data set contains 3 classes of 50 instances each, where each class refers to
a
type of iris plant. One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. dropdown:: References

  - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
    Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
    Mathematical Statistics" (John Wiley, NY, 1950).

```
  - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysi
s.
    (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
  - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
    Structure and Classification Rule for Recognition in Partially Exposed
    Environments".  IEEE Transactions on Pattern Analysis and Machine
    Intelligence, Vol. PAMI-2, No. 1, 67-71.
  - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactio
ns
    on Information Theory, May 1972, 431-433.
  - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
    conceptual clustering system finds 3 classes in the data.
  - Many, many more ...
```

Exemple : exploration

In [34]:
```
print(f'Noms des attributs : {iris.feature_names}')
```

```
Noms des attributs : ['sepal length (cm)', 'sepal width (cm)', 'petal length
(cm)', 'petal width (cm)']
```

In [35]:
```
print(f'Noms des classes : {iris.target_names}')
```

```
Noms des classes : ['setosa' 'versicolor' 'virginica']
```

In [36]:
```
print(f'Forme des données : {iris.data.shape}')
```

```
Forme des données : (150, 4)
```

In [37]:
```
print(f'Forme de la cible : {iris.target.shape}')
```

```
Forme de la cible : (150,)
```

Exemple : utilisation de `pandas`

In [38]:
```
import pandas as pd

# Créer un DataFrame

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target_names[iris.target]
```

Exemple : utilisation de `pandas` (suite)

In [39]:
```
# Afficher les premières lignes du DataFrame

print(df.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
\
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

  species
0  setosa
1  setosa
2  setosa
3  setosa
4  setosa
```

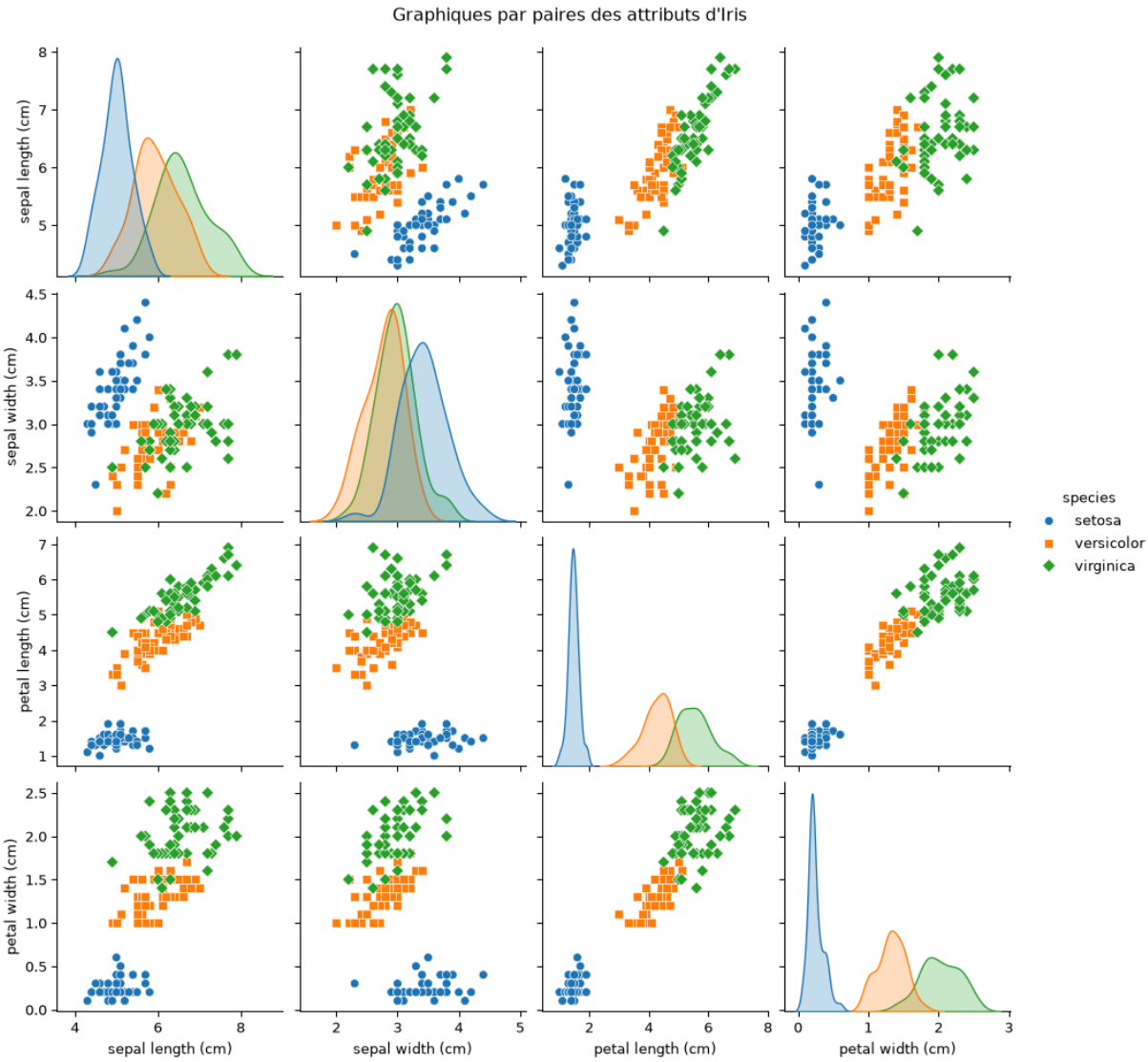
Exemple : utilisation de `pandas` (suite)

In [40]:
```python
# Statistiques descriptives

print(df.describe())
```

```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)
count        150.000000
mean           1.199333
std            0.762238
min            0.100000
25%            0.300000
50%            1.300000
75%            1.800000
max            2.500000
```

Exemple : utilisation de `seaborn`

In [41]:
```python
import seaborn as sns

# Graphiques par paires avec seaborn

sns.pairplot(df, hue='species', markers=["o", "s", "D"])
plt.suptitle("Graphiques par paires des attributs d'Iris", y=1.02)
plt.show()
```



Quelles perspectives peut-on tirer de l'examen des graphiques?

La figure présente les graphiques par paires des attributs d'Iris; la diagonale montre la distribution de chaque attribut. Chaque point représente un exemple (x), et sa couleur indique son étiquette (y).

Considérons d'abord les éléments diagonaux :

- Est-il possible de classer les exemples en utilisant un seul attribut ?
- Nous observons que la **longueur** ou la **largeur** du **sépale** seule ne peut pas distinguer les classes.
- Cependant, la **longueur du pétale** et la **largeur du pétale** permettent de distinguer **setosa** des deux autres espèces, même si elles ne séparent pas parfaitement **versicolor** et **virginica**.

La classe **setosa** forme fréquemment un groupe distinct.

Exemple : jeux d'entraînement et de test

In [42]:
```python
from sklearn.model_selection import train_test_split

# Diviser les données en jeux d'entraînement et de test

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=7,
    stratify=y,
)
```

Notre premier arbre de décision a été construit et évalué sur l'ensemble du jeu de données. Il peut très bien s'ajuster à ces exemples, mais cela ne nous indique pas avec quelle exactitude il prédira des **données inédites**.

Nous aurons beaucoup plus à dire sur les tests dans les semaines à venir.

Pour simplifier le récit, cette annexe explore l'ensemble du jeu de données avant d'introduire la séparation. Dans une expérience rigoureuse, le jeu de test doit être mis de côté avant de faire des choix de modélisation guidés par les données, puis demeurer intact jusqu'à l'évaluation finale.

`random_state` rend la partition pseudoaléatoire reproductible. En changeant sa valeur, on change les exemples réservés au test et, possiblement, l'exactitude mesurée. `stratify=y` conserve approximativement les proportions des classes dans les deux sous-ensembles.

Pratique : Essayez différentes valeurs de `random_state`. Qu'observez-vous, et pourquoi?

Exemple : création d'un nouvel arbre

In [43]:
```python
clf = tree.DecisionTreeClassifier(criterion="entropy", random_state=42)
```

Exemple : entraînement du modèle

In [44]:
```python
# Entraîner le modèle
clf.fit(X_train, y_train)
```

Exemple : prédictions

In [45]:
```python
# Faire des prédictions
y_pred = clf.predict(X_test)
```

Exemple : mesure de la performance

In [46]:
```python
from sklearn.metrics import classification_report, accuracy_score
# Faire des prédictions

# Évaluer le modèle
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(
    y_test,
    y_pred,
    labels=clf.classes_,
    target_names=iris.target_names[clf.classes_],
)

print(f'Exactitude : {accuracy:.2f}')
print('Rapport de classification :')
print(report)
```

```
Exactitude : 0.93
Rapport de classification :
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.90      0.90      0.90        10
   virginica       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```

Voici une discussion sur la persistance des modèles pour les modèles scikit-learn.

Marcel **Turcotte**

Marcel.Turcotte@uOttawa.ca

École de **science informatique** et de génie électrique (**SI**GE)

Université d'Ottawa