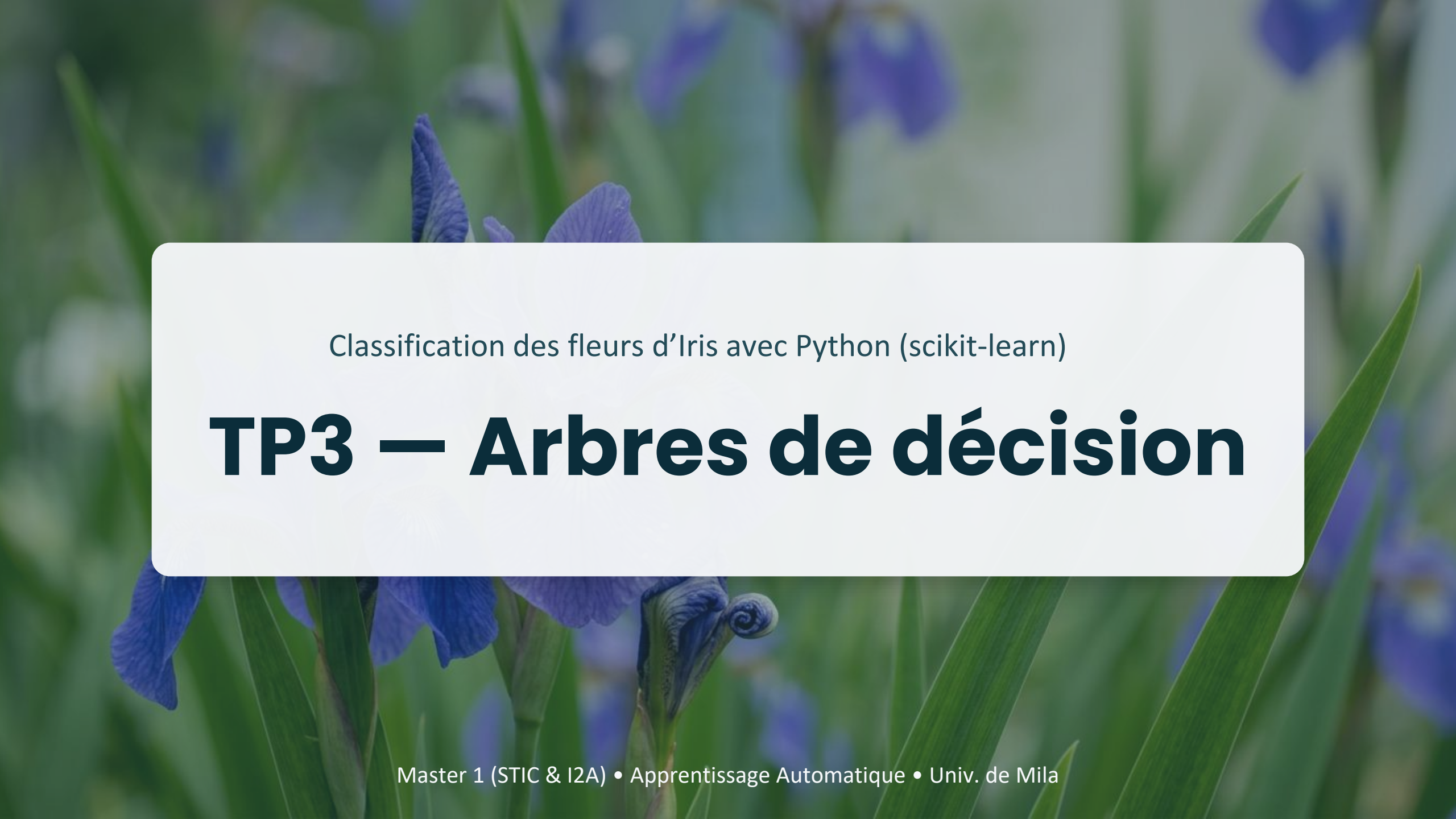


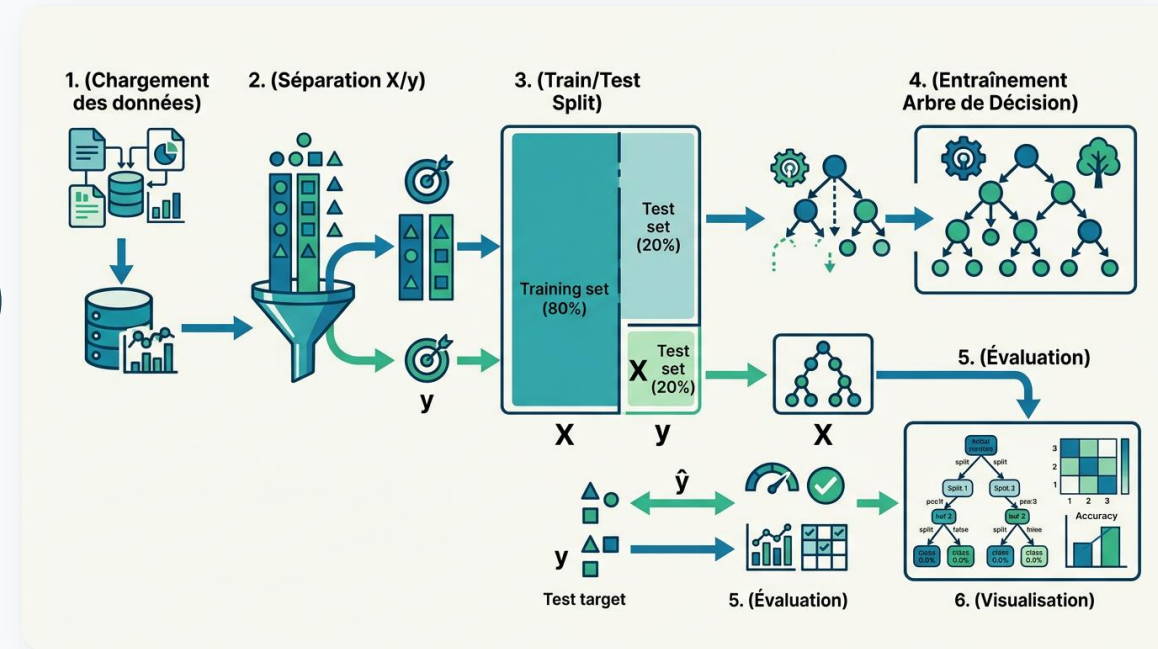


Classification des fleurs d'Iris avec Python (scikit-learn)

TP3 — Arbres de décision

Objectifs du TP

- **Chargement** : Importer le dataset Iris
- **Préparation** : Séparer X (attributs) et y (classes)
- **Partitionnement** : Train/Test Split (stratifié)
- **Modélisation** : Entraîner DecisionTreeClassifier
- **Évaluation** : Mesurer l'accuracy
- **Export** : Visualiser l'arbre (graphique et texte)



Pipeline classique de Machine Learning

Le Dataset IRIS

Caractéristiques clés :

- Échantillons : **150** (50 par classe)
- Classes : *Setosa*, *Versicolor*, *Virginica*
- Features : 4 mesures (Longueur/Largeur)
- Format : Numérique continu (cm)



Chargement des données

```
from *****import load_iris
import *** as pd
import ... as np
data = *****
df = pd.DataFrame(..., ...)
# Ajouter les noms des espèces
```

Alternative Excel : Utiliser `pd.read_excel('iris.xlsx')` si le dataset est fourni en fichier.

Chargement des données

```
#replace this with the actual names  
target = np.unique(data.target)  
target_names = np.unique(data.target_names)  
targets = dict(zip(target, target_names))  
df['Species'] = df['Species'].replace(targets)
```

Chargement des données



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

Séparer X et y

X : Variables d'entrée

Les caractéristiques (Features)

```
x = df.drop(columns="Species")
```

y : Variable de sortie

La cible à prédire (Target)

```
y = df["Species"]
```

On stocke aussi `feature_names` et `Labels` pour la visualisation finale

```
feature_names = X.columns
```

```
Labels = y.unique().
```

Train / Test Split

ENSEMBLE D'APPRENTISSAGE (TRAIN) — 70%



TEST — 30%



```
from sklearn.model_selection import *****  
X_train, X_test, y_train, y_test = *****(*, **, ***, *****)
```

Astuce : L'option *stratify=y* assure que les proportions de chaque espèce d'Iris sont respectées dans les deux ensembles.

Élaboration du modèle

```
from sklearn.tree import DecisionTreeClassifier
# Créer et entraîner le modèle
clf = ***** (criterion='gini', max_depth=3)
clf.fit(*****)
# Faire des prédictions
predicted = clf.predict(*****)
```

Critère Gini : Mesure l'impureté des nœuds.

Max Depth : Limite la profondeur pour éviter le *overfitting* (sur-apprentissage).

L'arbre de décision divise l'espace des données de manière récursive en utilisant les attributs les plus discriminants à chaque étape.

Élaboration du modèle

```
from sklearn.metrics import accuracy_score  
accuracy = *****(..., ...)  
print(f'Accuracy: {...}')
```

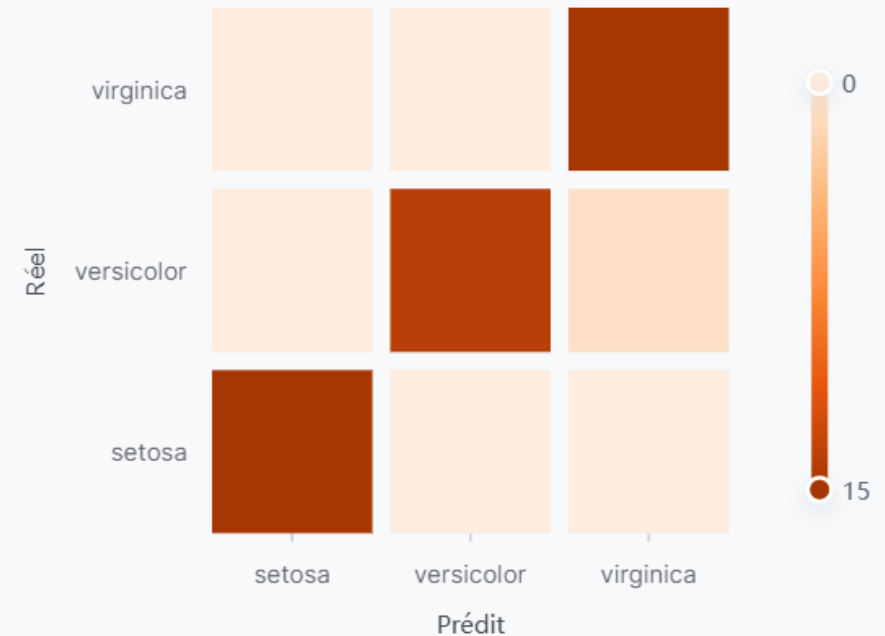
Analyse :

- Setosa : 100% correct
- Versicolor : 1 erreur (confusion avec Virginica)
- Virginica : 100% correct

Modèle très performant sur les classes bien séparées.

Accuracy
Score

98.8%



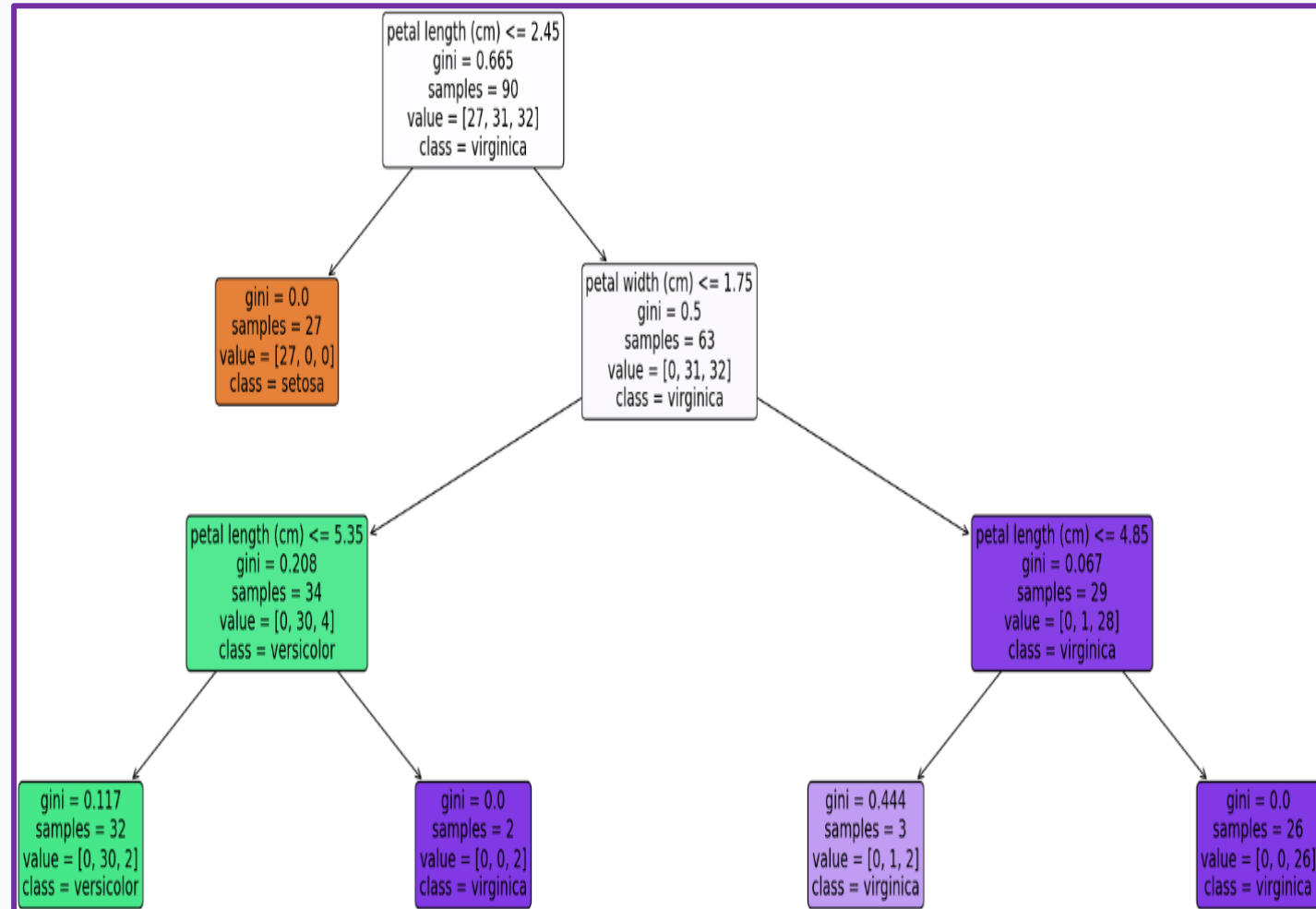
Visualisation de l'arbre

```
from sklearn import tree
import matplotlib.pyplot as plt

#plt the figure, setting a black background
plt.figure(figsize=(30,10), facecolor='w')

#create the tree plot
a = tree.plot_tree(...,
                    #use the feature names stored
                    feature_names = ...,
                    #use the class names stored
                    class_names = ...,
                    rounded = True,
                    filled = True,
                    fontsize=14)

#show the plot
plt.show()
```



Lecture d'un nœud : Chaque nœud indique la condition (*petal length* <= 2.45), l'impureté (*Gini*), le nombre d'échantillons et la classe majoritaire.

Export sous forme de règles

```
|--- petal length (cm) <= 2.45
| |--- class: setosa
|--- petal length (cm) > 2.45
| |--- petal width (cm) <= 1.55
| | |--- petal length (cm) <= 4.95
| | | |--- class: versicolor
| | |--- petal length (cm) > 4.95
| | | |--- class: virginica
| |--- petal width (cm) > 1.55 ...
```

```
#import relevant functions
from sklearn.tree import export_text

#export the decision rules
tree_rules = ... (clf, feature_names = list(feature_names))

#print the result
print(...)
```

L'utilisation de `export_text()` permet d'obtenir une version textuelle compacte, idéale pour documenter les décisions du modèle sans graphique.

À retenir & Perspectives

Interprétabilité

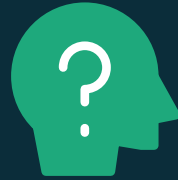
L'arbre est un modèle "boîte blanche". Chaque décision est explicable par des seuils sur les features.

Contrôle

Les hyperparamètres (*max_depth*) sont essentiels pour limiter le sur-apprentissage.

Extensions

Tester la validation croisée ou des méthodes d'ensemble (Random Forest) pour plus de robustesse.



Merci pour votre attention !

Avez-vous des questions ?