TP3 : Les arbres de décision

Selladurai Gowshigan

Introduction

Les arbres de décision sont des méthodes d'apprentissage non paramétriques, applicables à la classification comme à la régression. Ils construisent un modèle sous la forme d'une succession de règles « if–then » réparties en nœuds, de sorte à séparer au mieux les classes (classification) ou à approximer une fonction réelle (régression). Leur principal avantage est la clarté (modèle en boîte blanche) et la simplicité d'utilisation, tandis que leur inconvénient majeur est le risque de sur-apprentissage (overfitting) lorsque l'arbre devient trop profond.

I. Arbres pour la classification

1. Présentation de l'exemple Iris

Le jeu de données Iris comprend 150 observations décrivant la morphologie de trois espèces d'iris : setosa, versicolor et virginica. Chaque ligne contient quatre variables explicatives (attributs) et une étiquette de classe (0, 1 ou 2).

Attributs:

* longueur de sépale (cm)
* largeur de sépale (cm)
* longueur de pétale (cm)
* largeur de pétale (cm)

Étiquette (classe) :

* 0 : Iris setosa
* 1 : Iris versicolor
* 2 : Iris virginica

Questions :

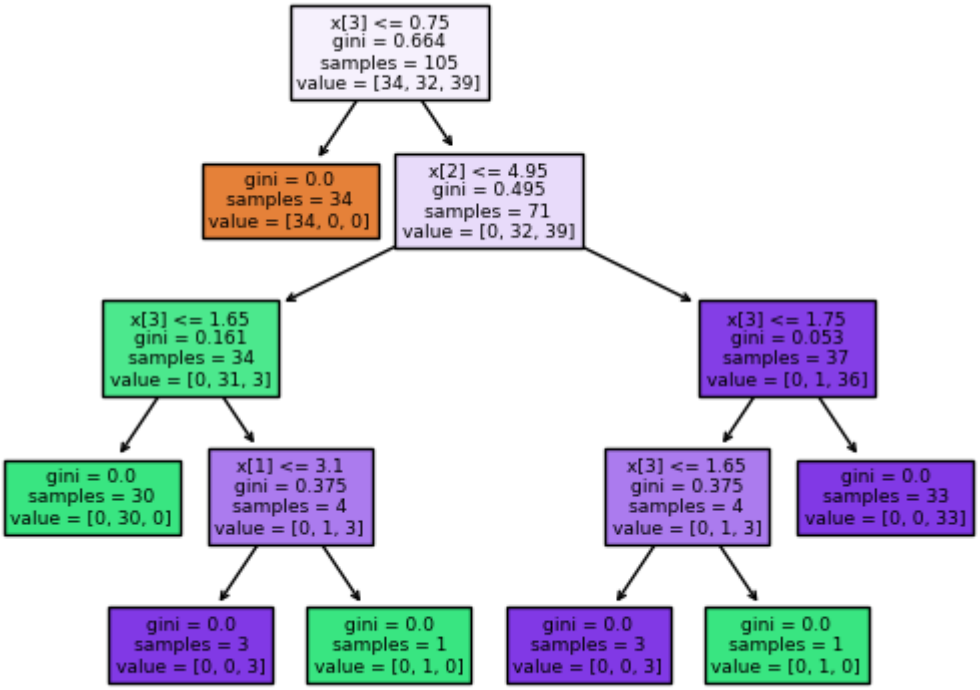
1. Calculer les statistiques (moyenne et écart-type) des quatre variables explicatives.

```
sepal length (cm) : Moyenne = 5.84, Écart-type = 0.83
sepal width (cm) : Moyenne = 3.06, Écart-type = 0.43
petal length (cm) : Moyenne = 3.76, Écart-type = 1.76
petal width (cm) : Moyenne = 1.20, Écart-type = 0.76
```

2. le nombre d'exemples par classe.

```
setosa : 50 exemples
versicolor : 50 exemples
virginica : 50 exemples
```

3. Arbre de décision (DecisionTreeClassifier) sur un jeu d'apprentissage (70 %), calculer la précision sur le test (30 %).



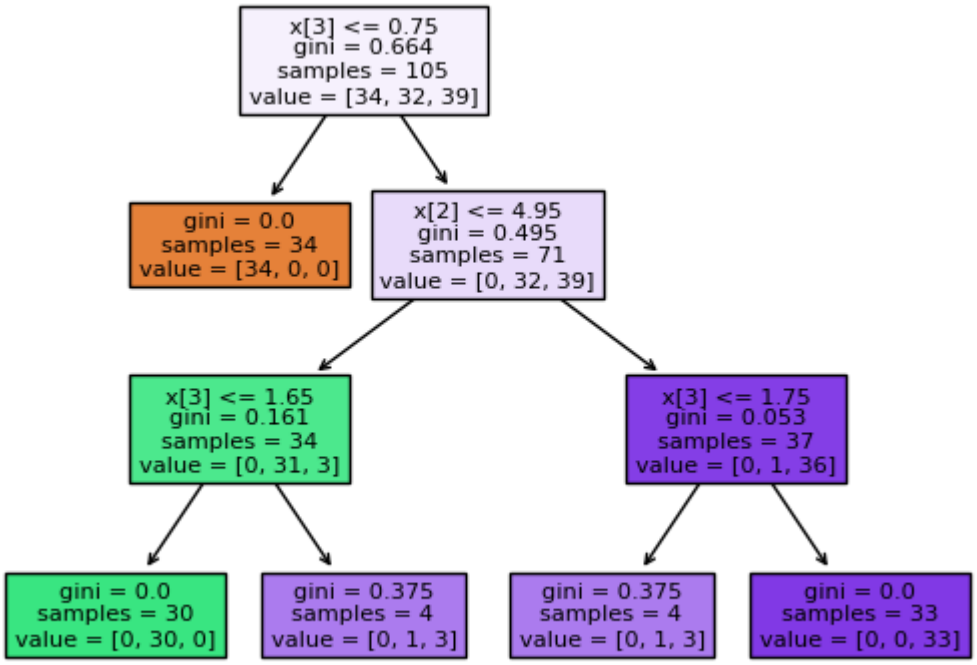
score : 0.9777777777777777

La précision est souvent très élevée (> 0.95), car Iris est relativement simple et bien séparé. Le graphique de l'arbre montre que, sans contrainte, l'arbre a plusieurs niveaux de profondeur pour séparer parfaitement les trois espèces.

4. Étudier l'impact des paramètres `max_depth` et `min_samples_leaf` sur la complexité de l'arbre et la performance.

- Pour éviter le sur-apprentissage et limiter la complexité, on change la valeurs du paramètre max_depth :
avec clf = tree.DecisionTreeClassifier(max_depth = 3)

```
clf = tree.DecisionTreeClassifier(max_depth = 3)
clf.fit(X_train, y_train)
tree.plot_tree(clf, filled=True)
clf.predict(X_test)
clf.score(X_test, y_test)
```

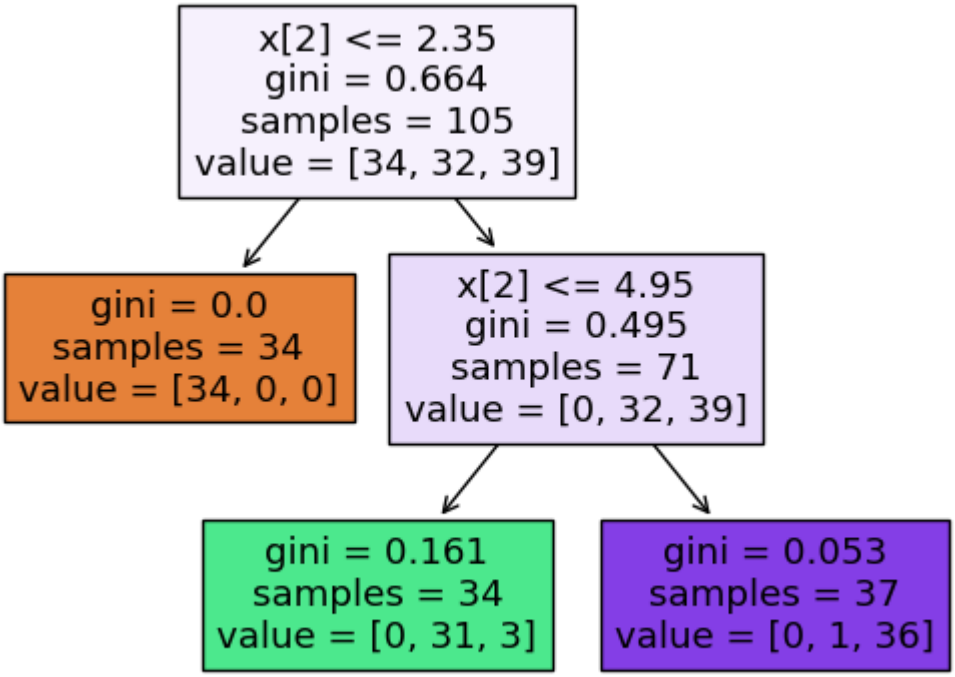


score : 0.9777777777777777

Lorsque `max_depth=3`, l'arbre est plus court : moins de règles « if–then ».
L'objectif est de montrer que réduire la profondeur peut suffire à obtenir un modèle plus simple tout en maintenant de bonnes performances.

L'autre paramètre est "min_samples_leaf", le nombre minimum d'exemplaires requis pour qu'un nœud soit une feuille.
-Si on change le min_samples_leaf :
avec clf = tree.DecisionTreeClassifier(min_samples_leaf = 20)

```
clf = tree.DecisionTreeClassifier(min_samples_leaf = 20)
clf.fit(X_train, y_train)
tree.plot_tree(clf, filled=True)
clf.predict(X_test)
clf.score(X_test, y_test)
```



score : 0.9111111111111111

Avec “min_samples_leaf=20”, chaque feuille doit contenir au moins 20 exemples ,
La précision sur le test chute légèrement ~ 0.91 vs ~ 0.98 pour l’arbre sans contrainte), car on perd en flexibilité.
On constate que plus on augmente “min_samples_leaf”, plus l’arbre risque d’être en sous-apprentissage mais il devrait être plus robuste aux bruits.

5. Redéfinition des jeux de données entrainement / test , avec 5% entraînement et 95 test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.05, random_state=0)

On ne garde que 5 % des 150 exemples (≈ 7 exemples) pour l’apprentissage,

```
#taux d'erreur = 1 - accuracy , ici fait par la méthode .score
taux_erreur = 1 - precision
print(f"Précision       : {precision:.4f}")
print(f"Taux d'erreur:  : {taux_erreur:.4f}")


Précision           : 0.7063
Taux d'erreur:      : 0.2937
```

Avec si peu de données d'apprentissage, l'arbre est peu performant : il sur-ajuste les 7 exemples, et ne généralise pas bien (≈ 70 % de bonnes prédictions sur le test).

---

6. Explorer une version « sous-échantillonnée » (5 % apprentissage, 95 % test) et optimiser "max_depth"/"min_samples_leaf" par "GridSearchCV".

Lorsque l'on dispose de très peu d'exemples pour l'apprentissage, il est important de trouver un compromis favorisant la généralisation. On réalise donc une recherche par grille sur les deux hyperparamètres principaux :

A noter : Comme X_train ne contient que 7 exemples un avertissement :

```
« The least populated class has ... members, less than n\_splits=4 »
```

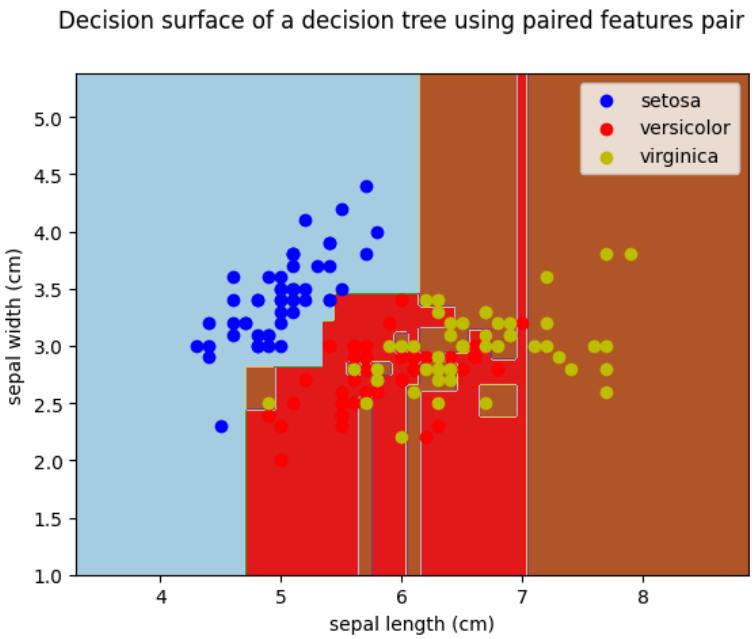
peut apparaître. Cela signifie que la validation croisée ne peut pas former plus de 4 plis pour chaque classe.

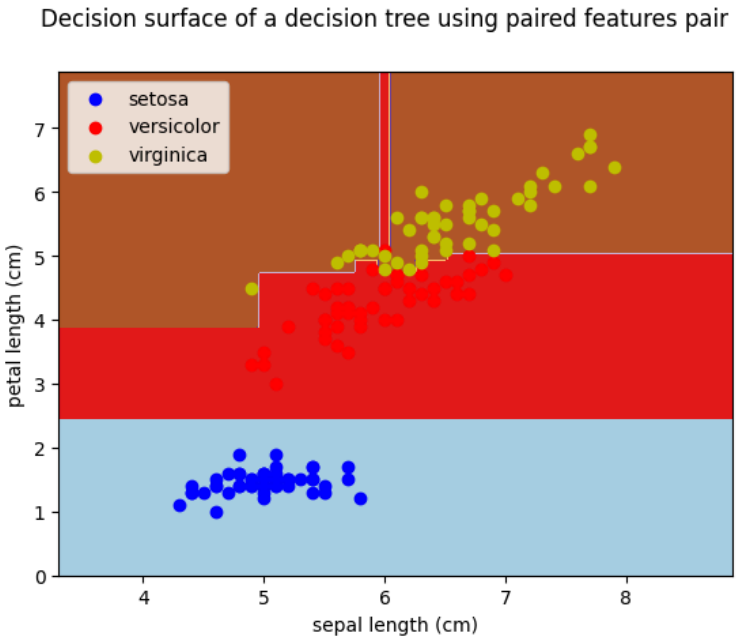
```
param_grid = {
      'max_depth': [None, 1, 2, 3, 4, 5],
      'min_samples_leaf': [1, 5, 10, 20, 50]
}
clf = tree.DecisionTreeClassifier(random_state=0)
grid_search = GridSearchCV(
      estimator=clf,
      param_grid=param_grid,
      scoring='accuracy',   # on optimise l'accuracy
      cv=4,                 # 5-fold cross-validation
      n_jobs=-1,            # parallélisation
      return_train_score=True
)
grid_search.fit(X_train, y_train)
print("Meilleurs paramètres :", grid_search.best_params_)
print(f"Meilleure accuracy CV : {grid_search.best_score_:.3f}")

Meilleurs paramètres : {'max_depth': None, 'min_samples_leaf': 1}
Meilleure accuracy CV : 0.625
```

On obtient le score moyen le plus élevé en CV (≈ 0.625).
Cela montre le manque de performance de la validation croisée sur très peu de données : avec aussi peu d'exemples, l'arbre sans contrainte finit par sur-ajuster chaque pli,

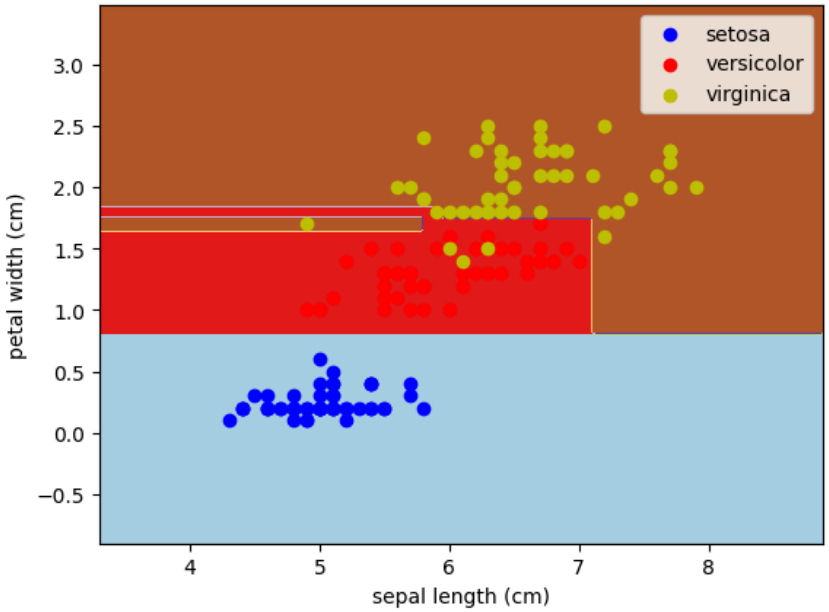
II. Affichage de la surface de décision

Pour chaque paire d'attributs (deux dimensions), on peut visualiser la zone de décision de l'arbre en affichant, pour chaque point d'une grille, la classe prédite. On garde ici uniquement deux colonnes de "iris.data".

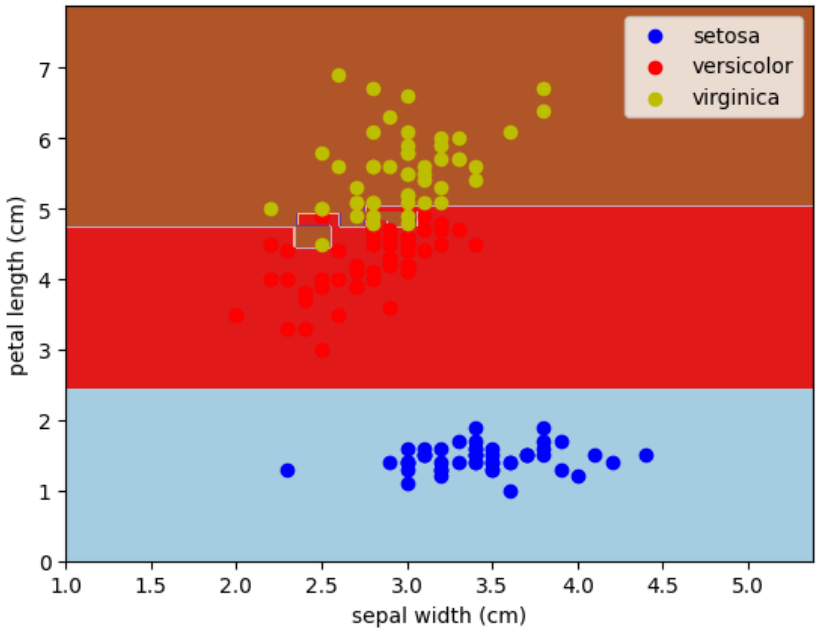


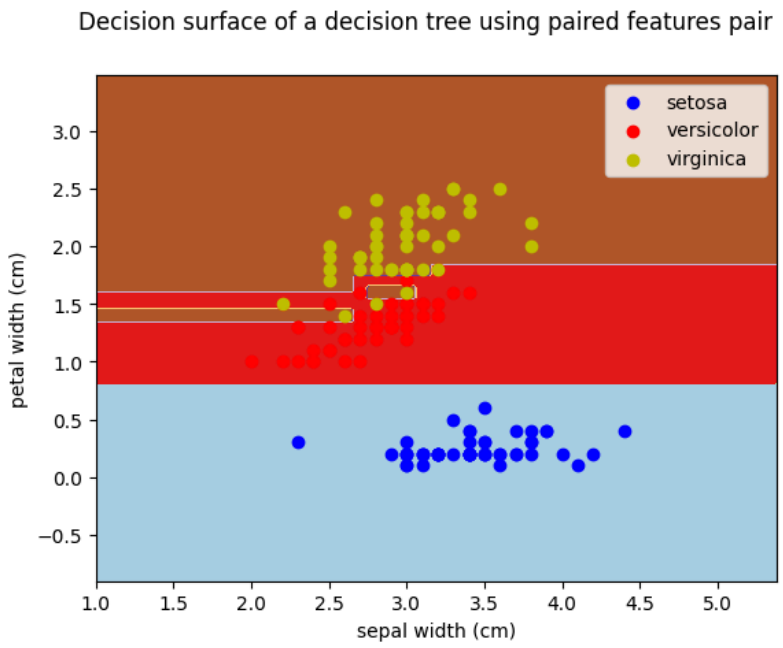


Decision surface of a decision tree using paired features pair



Decision surface of a decision tree using paired features pair





Observations:

Les surfaces sont souvent plus « nettes » sur les paires d'attributs qui distinguent le mieux les classes.
pour les pair (0 - 1) et (1 - 2)

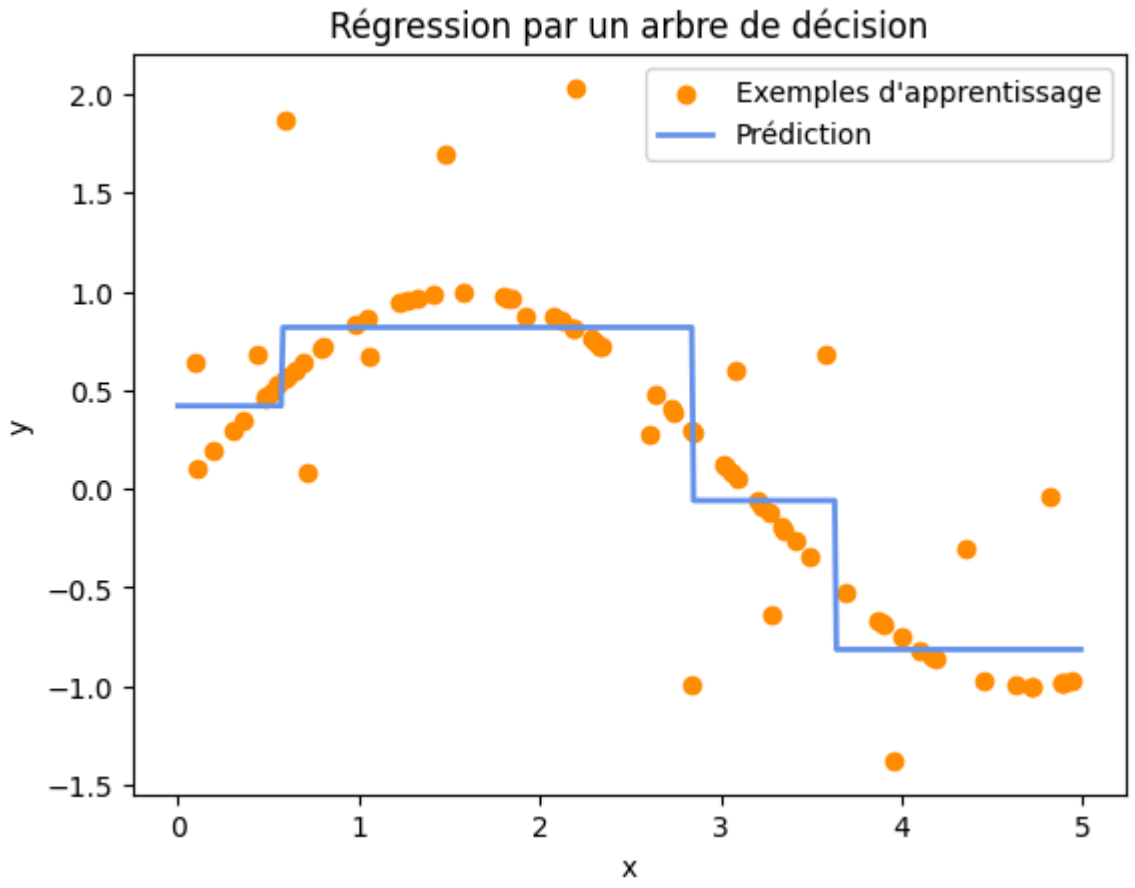
III. Arbres de décision pour la régression

Changer la valeur du paramètre max_depth. Que se passe-t-il si on prend une valeur trop grande ? Trop petite ?

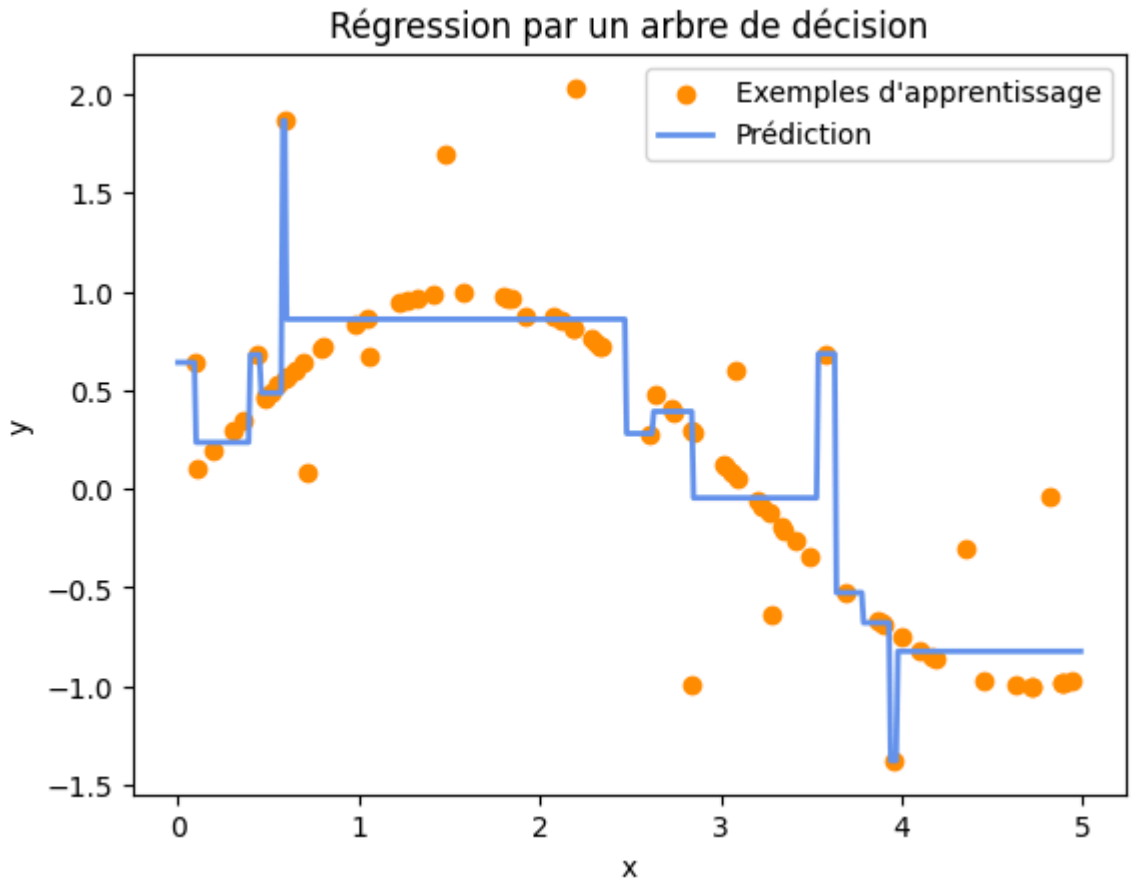
Si on définit un max_depth trop petit, le modèle est en underfitting , sous-apprentissage, il semble pas performer corréctement,

```
# Apprendre le modèle
reg = tree.DecisionTreeRegressor(max_depth=2)
reg.fit(X, y)
# Prédiction sur la même plage de valeurs
X_test = np.arange(0.0, 5.0, 0.01)[:, np.newaxis]
y_pred = reg.predict(X_test)
# Affichage des résultats
plt.figure()
plt.scatter(X, y, c="darkorange", label="Exemples d'apprentissage")
plt.plot(X_test,y_pred,color="cornflowerblue",label="Prédiction",linewid
th=2)
```

```
plt.xlabel("x")
plt.ylabel("y")
plt.title("Régression par un arbre de décision")
plt.legend()
plt.show()

Exemple avec max_depth = 1 :
```



Si on définit à l'inverse une valeur trop grande , le modèle est en overfitting, sur-apprentissage et il devient sensible aux bruits.
Exemple avec max_depth = 4



Changer le taux d'éléments affectés par le bruit (le y[::5]). Quand tous les éléments sont affectés par le bruit, faut-il préférer une valeur élevée ou faible pour max_depth?

```
X1 = [[0, 0], [2, 2]]
y1 = [0.5, 2.5]
clf = tree.DecisionTreeRegressor()
clf = clf.fit(X1, y1)
clf.predict([[1, 1]])

X1 = np.sort(5 * np.random.rand(80, 1), axis=0)
y1 = np.sin(X1).ravel()
# Apprendre le modèle
reg = tree.DecisionTreeRegressor(max_depth=4)
reg.fit(X1, y1)

# On ajoute un bruit aléatoire tous les échantillons
y1 += 3 * (0.5 - np.random.rand(80))

# Prédiction sur la même plage de valeurs
X_test1 = np.arange(0.0, 5.0, 0.01)[:, np.newaxis]
```

```
y_pred1 = reg.predict(X_test1)
# Affichage des résultats
plt.figure()
plt.scatter(X1, y1, c="darkorange", label="Exemples d'apprentissage")
plt.plot(X_test1,y_pred1,color="cornflowerblue",label="Prédiction",linew
idth=2)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Régression par un arbre de décision")
plt.legend()
plt.show()
```

Quand tous les éléments sont affectés par le bruit, une valeur faible pour max_depth performe une meilleure prédiction

Pour approfondir, chargez la base de données Diabètes du module sklearn.datasets et faire une partition aléatoire en partie apprentissage et partie test (70% apprentissage, 30% test). Construire un modèle d'arbre de regression sur cette base. Calculer l'erreur quadratique moyenne sur l'ensemble de test. Faire un grid search pour trouver la valeur du paramètre max_depth qui minimise cette erreur.

```
param_grid = {
      'max_depth': [None, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
}

grid_search = GridSearchCV(
      estimator=reg,
      param_grid=param_grid,
      scoring='neg_mean_squared_error',  # on optimise le mean squared
error
      cv=4,                    # 5-fold cross-validation
      n_jobs=-1,               # parallélisation
      return_train_score=True
)
grid_search.fit(X_train, y_train)

print("Meilleurs paramètres :", grid_search.best_params_)
print(f"Meilleure mse ( le plus faible) CV :
```

```
{-grid_search.best_score_:.3f}")



Après l'exécution de notre code , on se retrouve avec pour :

Meilleurs paramètres : {'max_depth': 2}
Meilleure mse ( le plus faible) CV : 4019.721
```

max_depth=2 minimise la MSE moyenne en validation croisée. Un arbre très peu profond (deux niveaux)
La MSE sur test obtenue basse que celle de l'arbre non contraint (~ 4000 vs ~ 6000).

IV. Conclusion générale

1. Classification Iris:

* Les arbres sans contrainte atteignent facilement > 95 % de précision sur le test.
* La limitation de la profondeur (`max_depth`) ou du nombre de feuilles (`min_samples_leaf`) permet de régulariser le modèle : réduire la profondeur ne dégrade pas forcément la performance sur Iris, mais simplifie l'arbre et augmente sa robustesse.
* Avec un sous-échantillonnage extrême (5 % apprentissage), l'arbre est très instable : on obtient \~ 70 % de bonnes prédictions et la validation croisée peine à trouver des paramètres vraiment significatifs.

2. Surface de décision :

Visualiser la frontière de décision pour chaque paire d'attributs permet de juger de la qualité de séparation des classes dans le sous-espace bidimensionnel.
Les paires les plus discriminantes sont généralement celles liées à la longueur/largeur des pétales (ou des sépales selon le TP), où la distribution des points s'organise en nuages relativement bien séparés.

3. Régression Diabetes :

L'arbre sans contrainte surapprend et présente une MSE élevée sur le test (\~ 6000).
En limitant la profondeur (`max_depth=2` d'après GridSearchCV), on diminue la MSE (\~ 4000) et on obtient un compromis plus satisfaisant.

Ce réglage montre l’importance de la régularisation dans les arbres de régression : une faible profondeur évite de coller à l’échantillon de bruit ou aux variations non pertinentes.