University Paris-Est Creteil, Creteil, France

Master 1 OIVM Optique Image Vision MultiMedia

Apprentissage supervisé
Les arbres de décision

Auteur:
Nassim Battache
Iskander Djouad

Superviseur :
Fekhreddine Keddous

March 29, 2022

Contents

Introduction **1**

1 Arbres pour la classification : **2**

2 Affichage de la surface de décision : **5**

3 Arbres de décision pour la régression : **9**

3.0.1 remarque 12

4 Conclusion : **14**

Introduction

Les arbres de décision sont des méthodes d'apprentissage non paramétriques utilisées pour des problèmes de classification et de régression. L'objectif est de créer un modèle qui prédit les valeurs de la variable cible, en se basant sur un ensemble de séquences de règles de décision déduites à partir des données d'apprentissage.

Chapter 1

Arbres pour la classification :

la definition d'un jeu de données minimaliste et la construction de l'arbre :

```
import numpy as np
from sklearn import tree
clf = tree.DecisionTreeClassifier()

X = [[0, 0], [1, 1]]
y = [0, 1]

clf = clf.fit(X, y)

clf.predict([[2., 2.]])

clf.predict_proba([[2., 2.]])
```

Figure 1.1: programme 1

Calcule de la moyenne des quatre variables: longueur de sépale, largueur de sépale, longueur de pétale et largeur de pétale.

```
import math
import statistics
import scipy
'''
pour visualiser la disrubusion de notre dataset
'''
print(scipy.stats.describe(iris.data[:,:3]) )
# X.columns=['Sepal_Length','Sepal_width','Petal_Length','Petal_width']
print( 'Sepal_Length mean :   ' + str(statistics.mean(X[0])))
print( 'Sepal_width mean :   ' + str(statistics.mean(X[1])))
print( 'Petal_Length mean :   ' + str(statistics.mean(X[2])))
print( 'Petal_width mean :   ' + str(statistics.mean(X[3])))
```

Figure 1.2: Données

le resulat est le suivant :

```
In [41]: runfile('/home/etudiant/untitled0.py', wdir='/home/etudiant')
DescribeResult(nobs=150, minmax=(array([4.3, 2. , 1. ]), array([7.9, 4.4, 6.9])),
mean=array([5.84333333, 3.05733333, 3.758     ]), variance=array([0.68569351, 0.18997942,
3.11627785]), skewness=array([ 0.31175306,  0.31576711, -0.27212767]), kurtosis=array([-0.57356795,
0.18097632, -1.39553589]))
Sepal_Length mean :  2.55
Sepal_width mean :  2.375
Petal_Length mean :  2.35
Petal_width mean :  2.35
```

Figure 1.3: Moyenne

Calcule de l'ecart type des quatre variables: longueur de sépale, largueur de sépale, longueur de pétale et largeur de pétale.

```
print( 'Sepal_Length ecartype :   ' + str(statistics.pstdev(X[0])))
print( 'Sepal_width ecartype :   ' + str(statistics.pstdev(X[1])))
print( 'Petal_Length ecartype :   ' + str(statistics.pstdev(X[2])))
print( 'Petal_width ecartype :   ' + str(statistics.pstdev(X[3])))
```

Figure 1.4: écart-type

le resulat est le suivant :

```
Sepal_Length ecartype :  1.8874586088176872
Sepal_width ecartype :  1.7640507362318127
Petal_Length ecartype :  1.7298843892006195
Petal_width ecartype :  1.6560495161679192
```

Figure 1.5: écart-type

le nombre d'exemples de chaque classe est :

```
print("le nombre d'exemple est "+str(np.bincount(iris.target)))
```

Figure 1.6: commande de nombre d'exemple

On remarque le nombre d'exemples de chaque classe est égal au nombre de cible.
Voici le resultat :

```
le nombre d'exemple est [50 50 50]
```

Figure 1.7: nombre d'exemple

le changements des valeurs de parametres max-depth et min-samples-leaf
Le problème ici étant particulièrement simple, refaites une division apprentissage/test avec 5

```
from sklearn import model_selection
X_train, X_test, y_train, y_test = model_selection.train_test_split(iris.data,
                                                                    iris.target,
                                                                    test_size=0.95,
                                                                    random_state=0)

for mdepth in [1, 2, 3, 4, 5]:
    clf = tree.DecisionTreeClassifier(max_depth=mdepth)
    clf = clf.fit(X_train, y_train)
    print("le score est de : "+str(clf.score(X_test, y_test)))

for msplit in [2, 5, 10, 20]:
    clf = tree.DecisionTreeClassifier(min_samples_split=msplit)
    clf = clf.fit(X_train, y_train)
    print("le score est de : "+str(clf.score(X_test, y_test)))

# clf = tree.DecisionTreeClassifier(max_depth = 3)
# clf = tree.DecisionTreeClassifier(min_samples_leaf = 20)
print("le score est de : "+str(clf.score(X_test, y_test)))
print("le teux d'erreur est de : "+str(1-(clf.score(X_test, y_test))))
```

Figure 1.8: nombre d'exemple

```
le score est de : 0.6573426573426573
le score est de : 0.8881118881118881
le score est de : 0.951048951048951
le score est de : 0.951048951048951
le score est de : 0.7062937062937062
le score est de : 0.7062937062937062
le score est de : 0.6223776223776224
le score est de : 0.32167832167832167
le score est de : 0.32167832167832167
```

Figure 1.9: nombre d'exemple

Voici le resultat :

Calculez le taux d'éléments mal classifiés sur l'ensemble de test. Faites varier (ou mieux, réalisez une recherche par grille avec GridSearchCV) les valeurs des paramètres max-depth et min-samples-leaf pour mesurer leur impact sur ce score.

```
from sklearn import model_selection
X_train, X_test, y_train, y_test = model_selection.train_test_split(iris.data,
                                                                    iris.target,
                                                                    test_size=0.30,
                                                                    random_state=0)

from sklearn.model_selection import GridSearchCV
pgrid = {"max_depth": [1, 2, 3, 4, 5, 6, 7],
      "min_samples_split": [2, 3, 5, 10, 15, 20]}
grid_search = GridSearchCV(tree.DecisionTreeClassifier(), param_grid=pgrid, cv=10)
grid_search.fit(X_train, y_train)
print(grid_search.best_estimator_.score(X_test, y_test))
```

Figure 1.10: le taux

Le résultat est le suivant :

```
le score est de : 0.32167832167832167
le teux d'erreur est de : 0.6783216783216783
0.9777777777777777
```

Figure 1.11: le taux d'erreur et le score

Chapter 2

Affichage de la surface de décision :

Refaire l'affichage pour les autres paires d'attributs. Sur quelle paire la séparation entre les classes est la plus marquée.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn import tree
iris = load_iris()
X, y = iris.data, iris.target
# Paramètres
n_classes = 3
plot_colors = "bry" # blue-red-yellow
plot_step = 0.02

# Choisir les attributs longueur et largeur des pétales
pair = [2, 3]

# On ne garde seulement les deux attributs
X = iris.data[:, pair]
y = iris.target

# Apprentissage de l'arbre
clf = tree.DecisionTreeClassifier().fit(X, y)

# Affichage de la surface de décision
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step), np.arange(y_min, y_max, plot_step))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)
plt.xlabel(iris.feature_names[pair[0]])
plt.ylabel(iris.feature_names[pair[1]])
plt.axis("tight")

# Affichage des points d'apprentissage
for i, color in zip(range(n_classes), plot_colors):
    idx = np.where(y == i)
    plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i], cmap=plt.cm.Paired)
plt.axis("tight")
plt.suptitle("Decision surface of a decision tree using paired features")
plt.legend()
plt.show()
```

Figure 2.1: l'affichage pour les autres paires d'attributs

voici le resultat suivant :

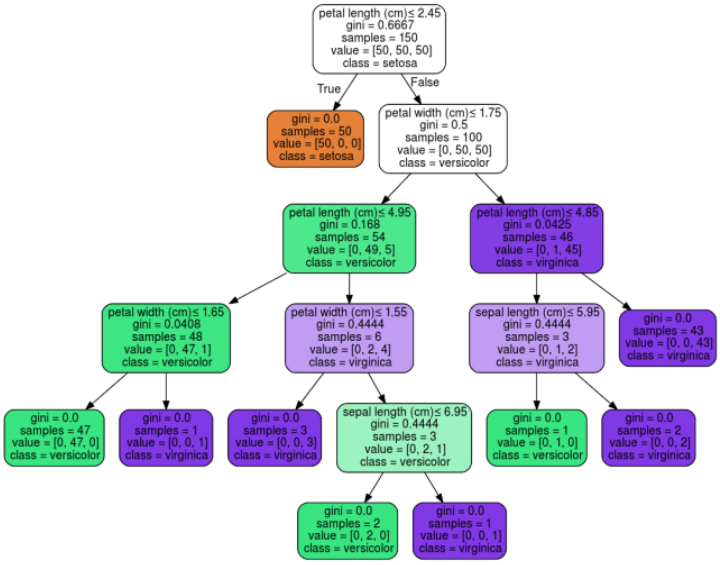


Figure 2.2: l'affichage pour les autres paires d'attributs

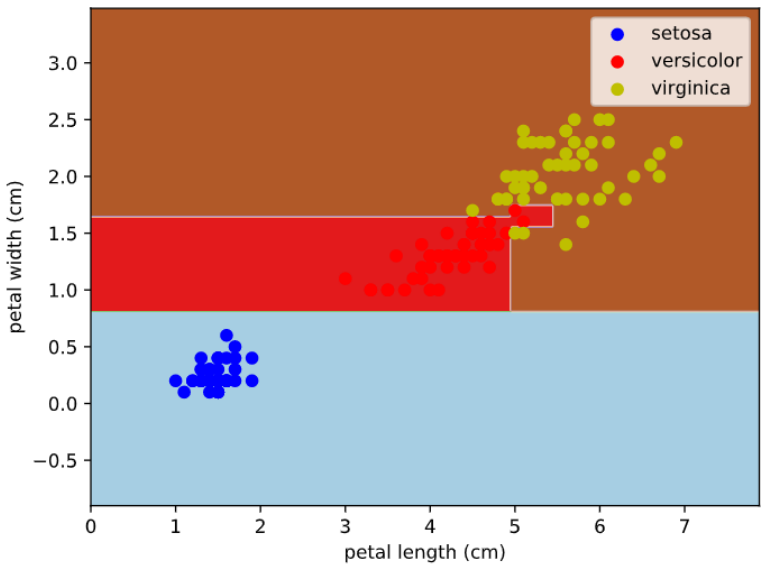


Figure 2.3: l'affichage pour les autres paires d'attributs

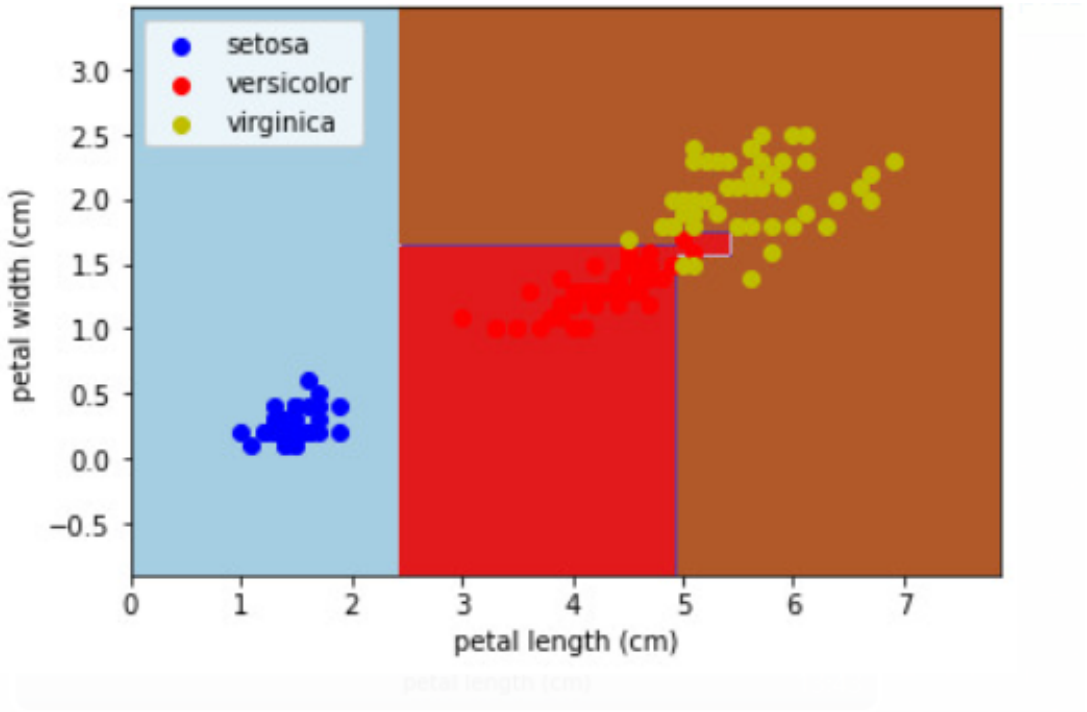


Figure 2.4: l'affichage pour les autres paires d'attributs

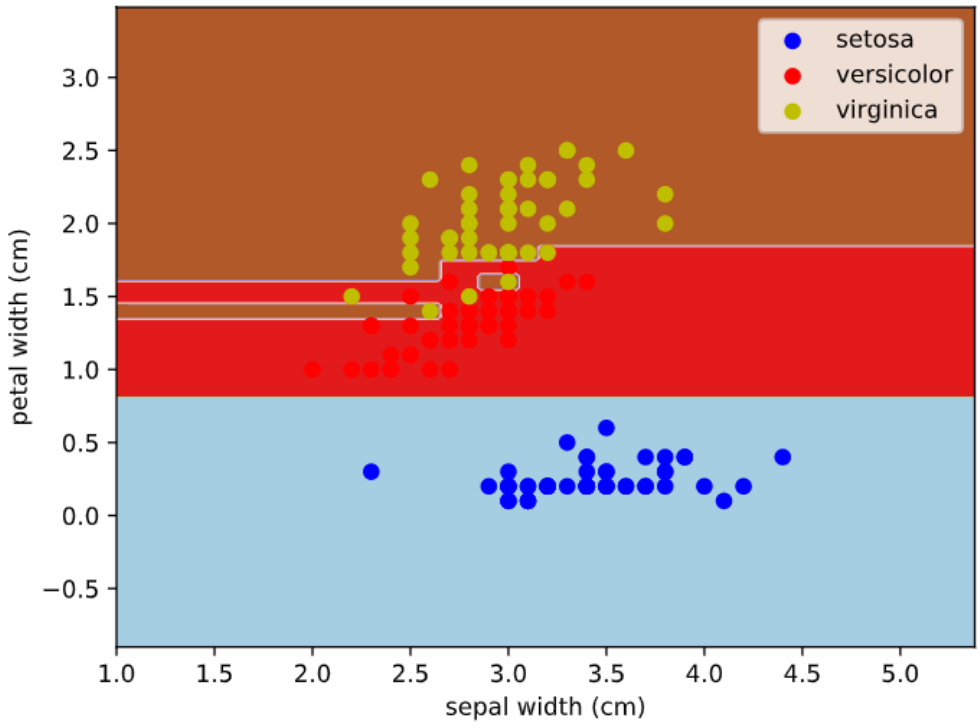


Figure 2.5: l'affichage pour les autres paires d'attributs

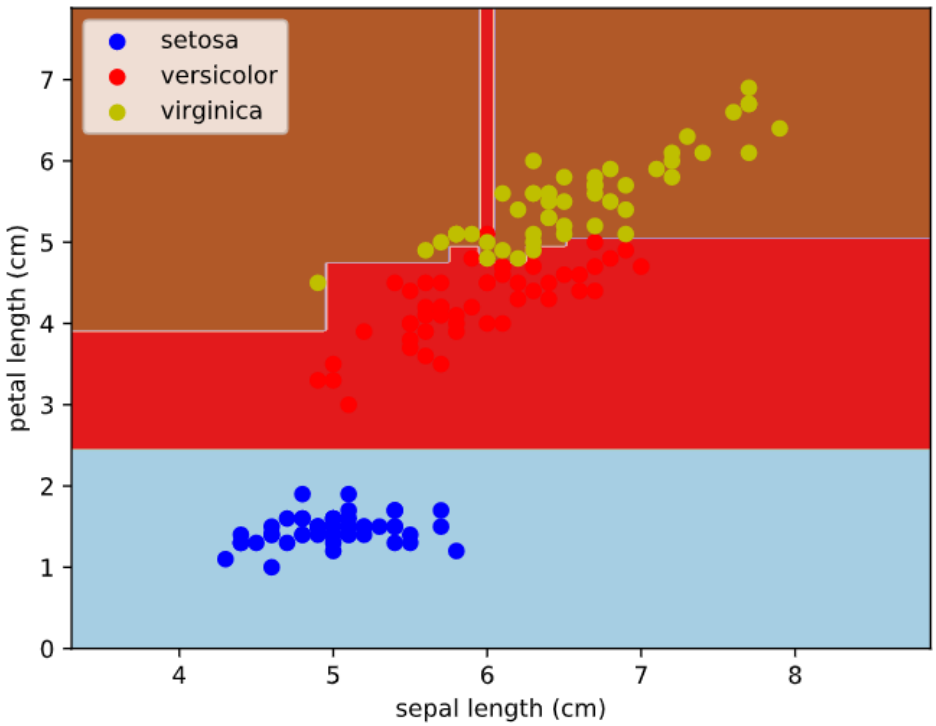


Figure 2.6: l'affichage pour les autres paires d'attributs

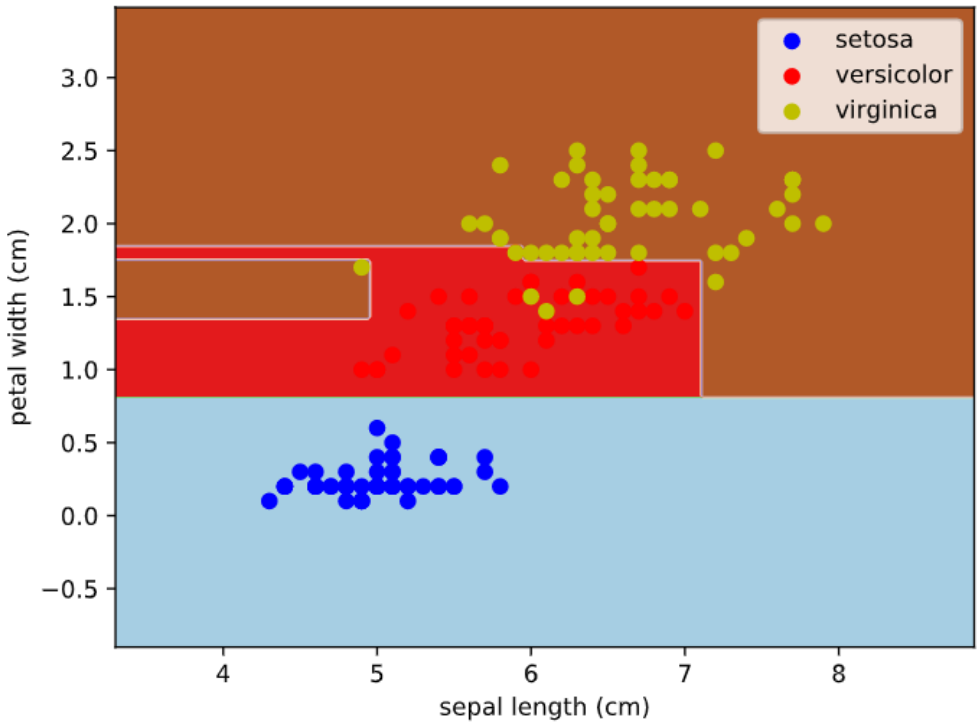


Figure 2.7: l'affichage pour les autres paires d'attributs

Chapter 3

Arbres de décision pour la régression :

Affichage de la surface de décision

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

```
from sklearn import tree
X = [[0, 0], [2, 2]]
y = [0.5, 2.5]
clf = tree.DecisionTreeRegressor()
clf = clf.fit(X, y)
clf.predict([[1, 1]])

import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
# Créer les données d'apprentissage
np.random.seed(0)
X = np.sort(5 * np.random.rand(80, 1), axis=0)
y = np.sin(X).ravel()

fig = plt.figure(figsize=(12, 4))
fig.add_subplot(121)
plt.plot(X, y)
plt.title("Signal sinusoïdal pur")
# On ajoute un bruit aléatoire tous les 5 échantillons
y[::5] += 3 * (0.5 - np.random.rand(16))
fig.add_subplot(122)
plt.plot(X, y)
plt.title("Signal sinusoïdal bruité")

# Apprendre le modèle
reg = DecisionTreeRegressor(max_depth=15
                            )
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",
linewidth=2)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Régression par un arbre de décision avec max_depth =15")
plt.legend()
plt.show()
```

Figure 3.1: programme 2

Voici les résultats obtenue:

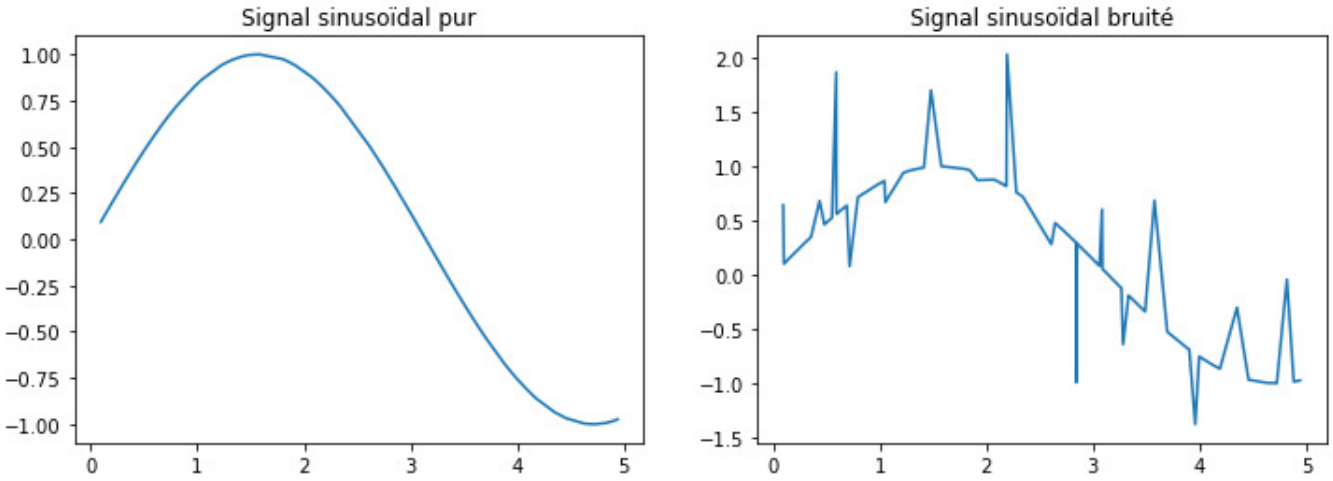


Figure 3.2: programme 2

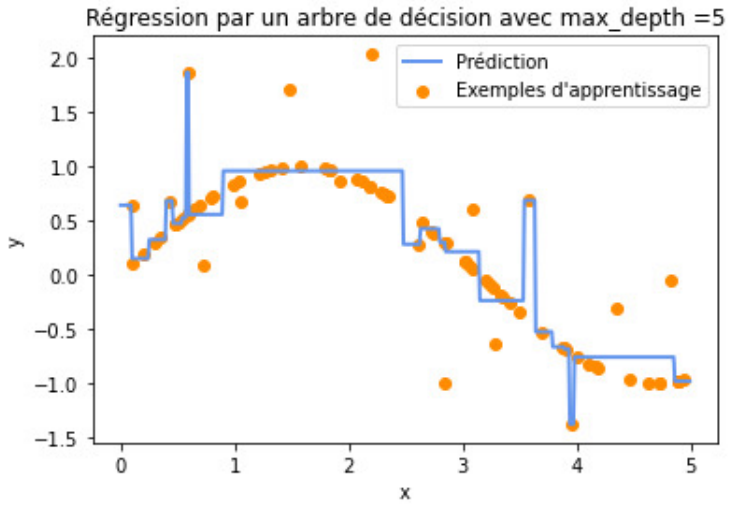


Figure 3.3: programme 2

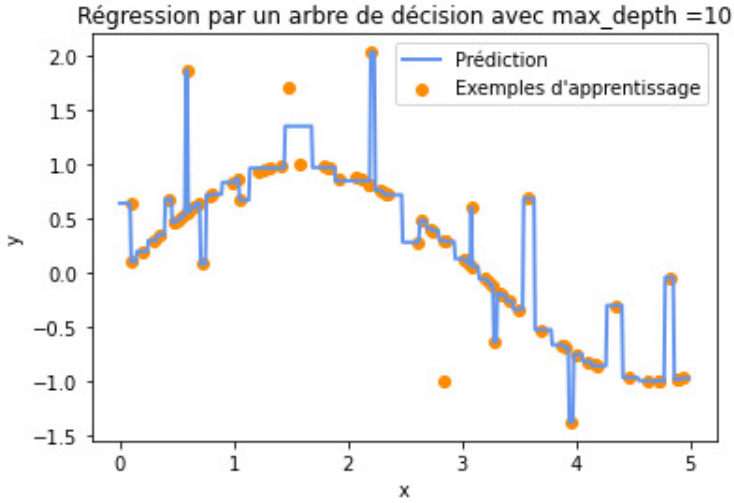


Figure 3.4: programme 2

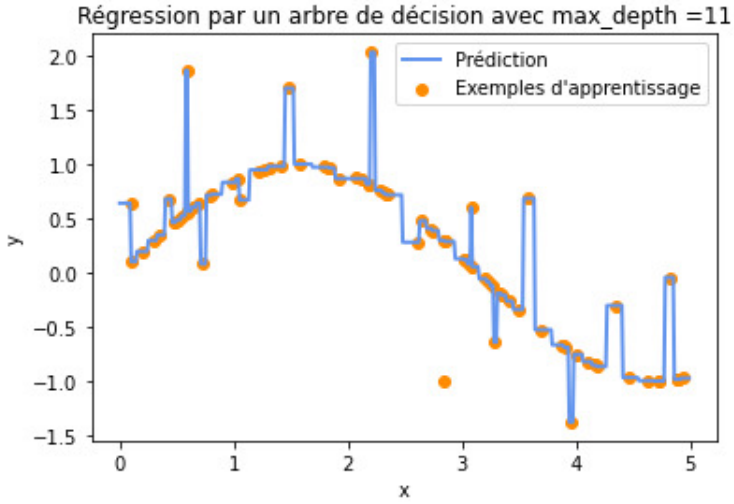


Figure 3.5: programme 2

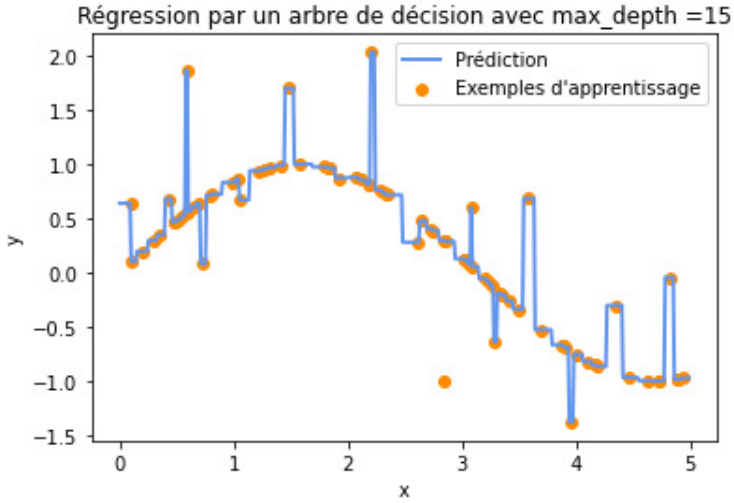
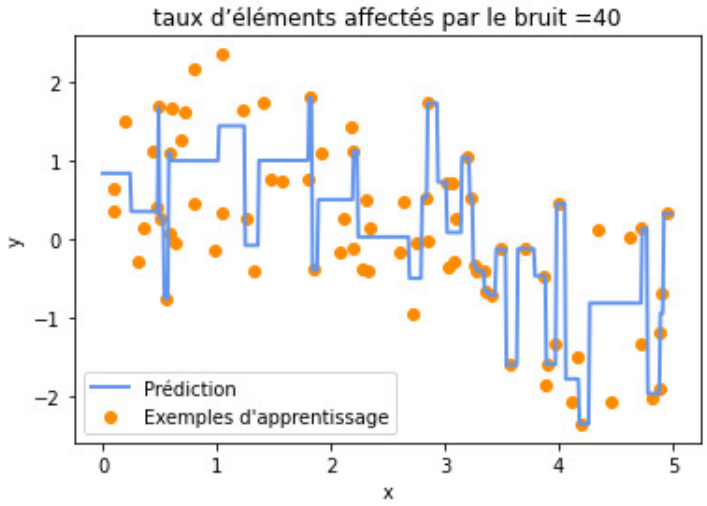
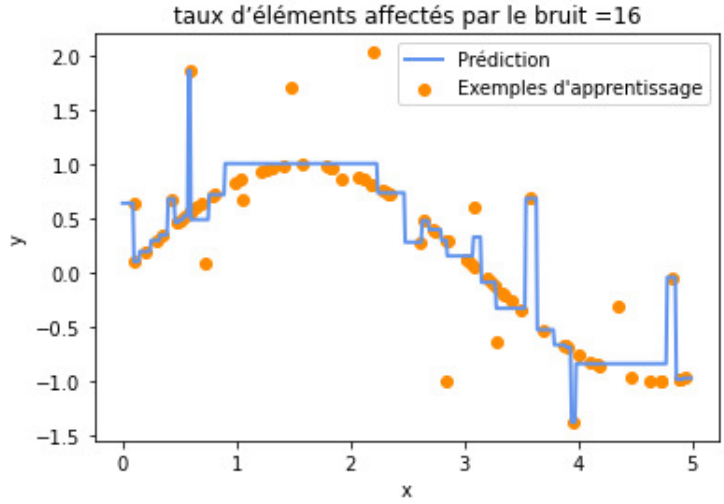
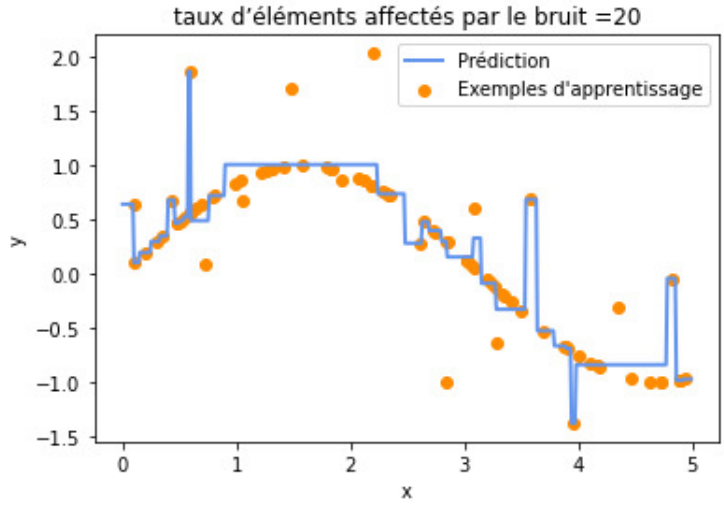
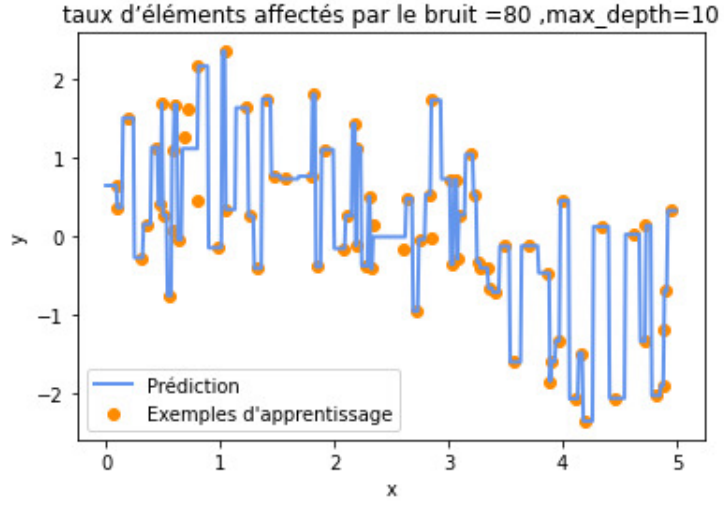


Figure 3.6: programme 2









3.0.1 remarque

Si max-depth est une valeur trop grande, le modèle suit le bruit, donc on a du sur-apprentissage. Si max-depth est une valeur trop petite, le modèle résultant ne suit pas bien les données,donc en est en sous-apprentissage. la généralisation est affecté. Il faut donc trouver un compromis en fonction du cout des erreurs .

Pour approfondir, chargez la base de données Diabètes du module sklearn.datasets et faire une partition aléatoire en partie apprentissage et partie test (70regression 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.

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn import model_selection
from sklearn.metrics import mean_squared_error
from sklearn.datasets import load_diabetes

diabetes = load_diabetes()
X_train, X_test, y_train, y_test = model_selection.train_test_split(diabetes.data, diabetes.target, test_size=0.30, random_state=0)

clf = DecisionTreeRegressor(max_depth=4)
clf.fit(X_train, y_train)
y_predicted = clf.predict(X_test)
mean_squared_error(y_test, y_predicted)
# --> 4365.3425018803518

pgrid = {"max_depth": [2, 3, 4, 5, 6, 7]}
grid_search = GridSearchCV(DecisionTreeRegressor(), param_grid=pgrid, scoring='neg_mean_squared_error', cv=10)
grid_search.fit(X_train, y_train)
y_predicted = grid_search.best_estimator_.predict(X_test)
print("l'erreur quadratique moyenne : "+str( mean_squared_error(y_test, y_predicted) ) )

print("la valeur du paramètre max_depth qui minimise cette erreur"+str(grid_search.best_params_))
```

voici le resultat :

```
l'erreur quadratique moyenne : 4029.0729322641987
la valeur du paramètre max_depth qui minimise cette erreur{'max_depth': 2}
```

Chapter 4

Conclusion :

Plus l'arbre généré est complexe, mieux le modèle « explique » les donnés d'apprentissage mais plus le risque de sur-apprentissage (over-fitting) est élevé.

Ils sont simples à comprendre et à visualiser.

Ils nécessitent peu de préparation des données (normalisation, etc.).

Le coût d'utilisation des arbres est logarithmique.

Ils sont capables d'utiliser des données catégorielles et numériques.

Ils sont capables de traiter des problèmes multi-classe.

Modèle en boîte blanche : le résultat est facile à conceptualiser et à visualiser