Rapport TP2

Paul Crespin

Classification avec les arbres

On commence par importer les librairies nécessaires.

```
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from matplotlib import rc

from sklearn import tree, datasets
from lib.tp_arbres_source import (rand_gauss, rand_bi_gauss, rand_tri_gauss,
                                  rand_checkers, rand_clown,
                                  plot_2d, frontiere)
```

Q1. Dans le cadre de la régression (i.e., quand on cherche à prédire une valeur numérique pour Y et non une classe), proposez une autre mesure d'homogénéité. Justifier votre choix.

Dans le cadre de la régression, une mesure d'homogénéité pouvant être utilisée est la somme des carrés résiduels (ou RSS) qui quantifie les erreurs résiduelles entre les valeurs prédites par le modèle de régression et les valeurs réelles de la variable dépendante, à prédire. Sa valeur est calculé avec la différence entre la valeur réelle et la valeur prédite par le modèle de régression pour chaque observation de l'ensemble de données puis, en ajoutant tous les carrés des résidus. Une RSS plus faible indiquera alors une meilleure homogénéité des résidus.

Avec scikit-learn, on peut construire des arbres de décision grâce au package tree. On obtient un classifieur avec tree.DecisionTreeClassifier.

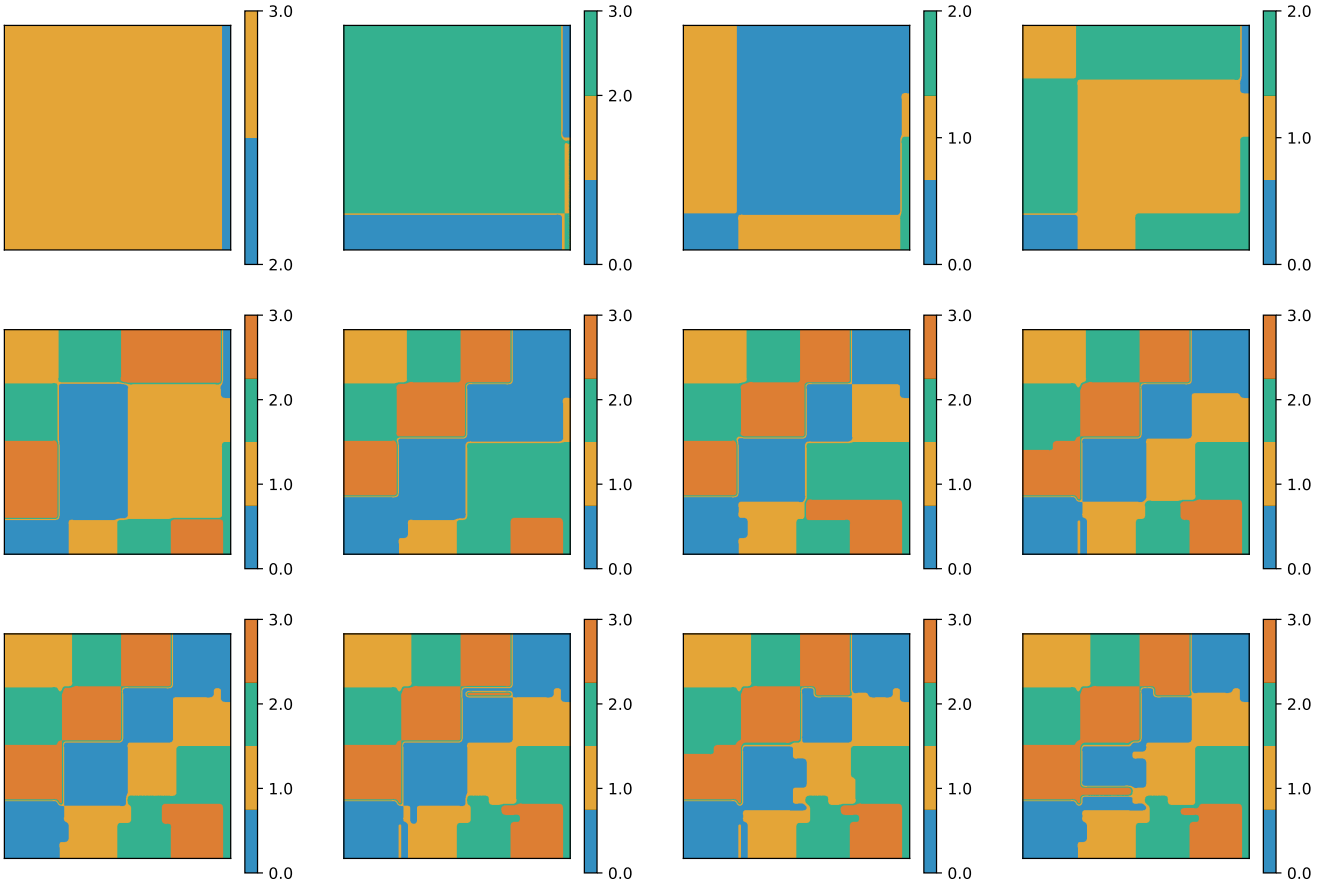
Q2. Simulez avec rand_checkers des échantillons de taille n = 456 (attention à bien équilibrer les classes). Créez deux courbes qui donnent le pourcentage d'erreurs commises en fonction de la profondeur maximale de l'arbre (une courbe pour Gini, une courbe pour l'entropie). On laissera les autres paramètres à leur valeurs par défaut.

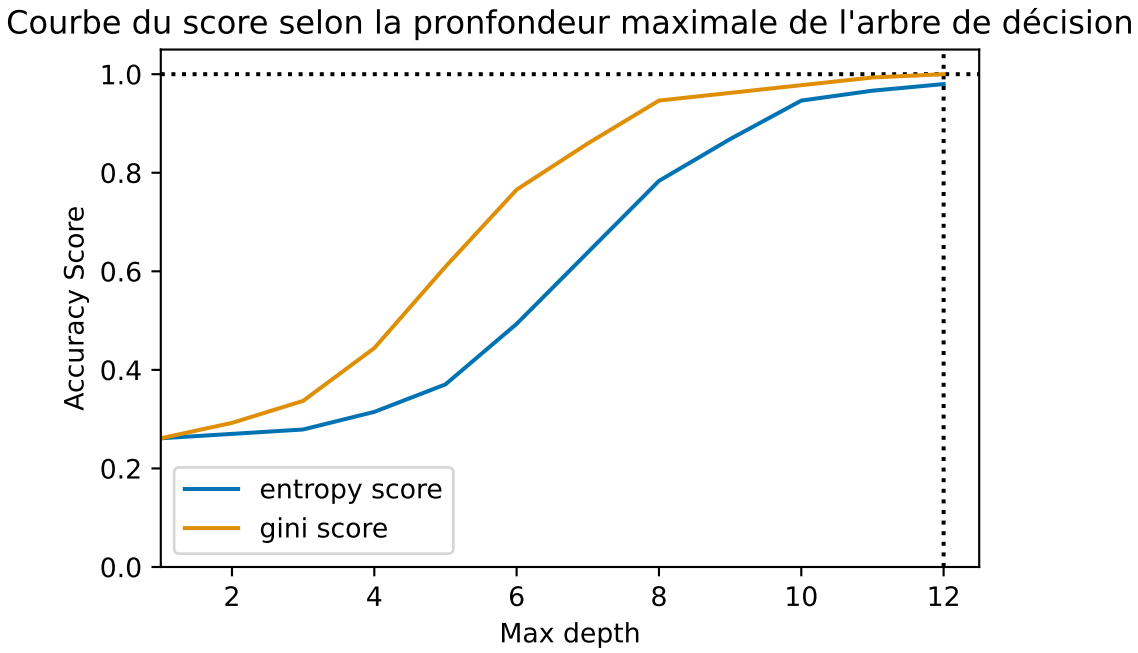
```
dt_entropy = tree.DecisionTreeClassifier(criterion = 'entropy')
dt_gini = tree.DecisionTreeClassifier() # par défaut, criterion = 'gini'

np.random.seed(10) # On fixe la seed.

n = 456
data = rand_checkers(n // 4, n // 4, n // 4, n // 4) #On crée notre jeu de données.
n_samples = len(data)
X = data[:, :2]
Y = data[:, 2]

dt_gini.fit(X, Y)
dt_entropy.fit(X, Y)

print("Gini criterion")
print(dt_gini.score(X, Y))

print("Entropy criterion")
print(dt_entropy.score(X, Y))

# Afficher les scores en fonction du paramètre max_depth

dmax = 12
scores_entropy = np.zeros(dmax)
scores_gini = np.zeros(dmax)
max_depth_entropy = 0
max_depth_gini = 0
max_score_entropy = 0
max_score_gini = 0

plt.figure(figsize=(15, 10))

for i in range(dmax):
    dt_entropy = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = i+1)
    dt_entropy.fit(X, Y)
    scores_entropy[i] = dt_entropy.score(X, Y)
    if scores_entropy[i] > max_score_entropy:
        max_score_entropy = scores_entropy[i]
        max_depth_entropy = i+1
```

```
    dt_gini = tree.DecisionTreeClassifier(max_depth = i+1)
    dt_gini.fit(X, Y)
    scores_gini[i] = dt_gini.score(X, Y)
    if scores_gini[i] > max_score_gini:
        max_score_gini = scores_gini[i]
        max_depth_gini = i+1

    plt.subplot(3, 4, i + 1)
    frontiere(lambda x: dt_gini.predict(x.reshape((1, -1))), X, Y, step=50, samples=False)

plt.draw()


plt.figure() # On plot le score des deux méthodes selon la profondeur.

plt.xlim([1, 12.5])
plt.ylim([0, 1.05])
plt.axvline(x = 12, color = 'black', linestyle = "dotted")
plt.axhline(y = 1, color = 'black', linestyle = "dotted")

plt.plot(list(range(1, dmax + 1)), scores_entropy, label = "entropy score")
plt.plot(list(range(1, dmax + 1)), scores_gini, label = "gini score")

plt.xlabel('Max depth')
plt.ylabel('Accuracy Score')
plt.title("Courbe du score selon la pronfondeur maximale de l'arbre de décision")
plt.legend()

plt.draw()

print("Scores with entropy criterion: ", scores_entropy[:5])
print(scores_entropy[5:])
print("Scores with Gini criterion: ", scores_gini[:5])
print(scores_gini[5:])
```

```
Gini criterion
1.0
Entropy criterion
1.0
Scores with entropy criterion:  [0.26116071 0.27008929 0.27901786 0.31473214 0.37053571]
[0.49330357 0.63839286 0.78348214 0.86830357 0.94642857 0.96651786
 0.97991071]
Scores with Gini criterion:  [0.26116071 0.29241071 0.33705357 0.44419643 0.609375  ]
[0.765625   0.859375   0.94642857 0.96205357 0.97767857 0.99330357
```

```
1.          ]
```

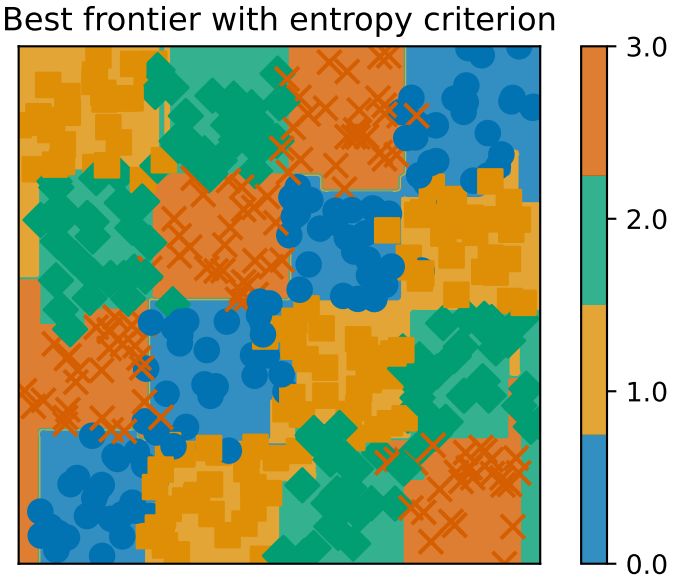

3.0
2.0
3.0
2.0
2.0
1.0
2.0
1.0
3.0
2.0
1.0
0.0
3.0
2.0
1.0
0.0
3.0
2.0
1.0
0.0
3.0
2.0
1.0
0.0
3.0
2.0
1.0
0.0
3.0
2.0
1.0
0.0
3.0
2.0
1.0
0.0
3.0
2.0
1.0
0.0
3.0
2.0
1.0
0.0



Q3. Afficher la classification obtenue en utilisant la profondeur qui minimise le pourcentage d'erreurs obtenues avec l'entropie (utiliser si besoin les fonctions plot_2d et frontiere du fichier source).

```
dt_entropy = tree.DecisionTreeClassifier(criterion = 'entropy',
max_depth = max_depth_entropy)
dt_entropy.fit(X, Y)

plt.figure()
frontiere(lambda x: dt_entropy.predict(x.reshape((1, -1))), X, Y, step=100)
plt.title("Best frontier with entropy criterion")
plt.draw()

print("Best scores with entropy criterion: ", dt_entropy.score(X, Y))
```

```
Best scores with entropy criterion:  0.9799107142857143
```


Best frontier with entropy criterion
3.0
2.0
1.0
0.0

Q4. Exporter la représentation graphique de l'arbre.

Voir https://scikit-learn.org/stable/modules/tree.html#classification

```
import graphviz
from sklearn.tree import export_graphviz

donnes = export_graphviz(dt_entropy)

graph = graphviz.Source(donnes)
#graph.render("./graphviz/entropy_tree", format='pdf')
# L'arbre est trop grand pour être affiché dans le rapport.
# Il est disponible dans le dossier graphviz.
```

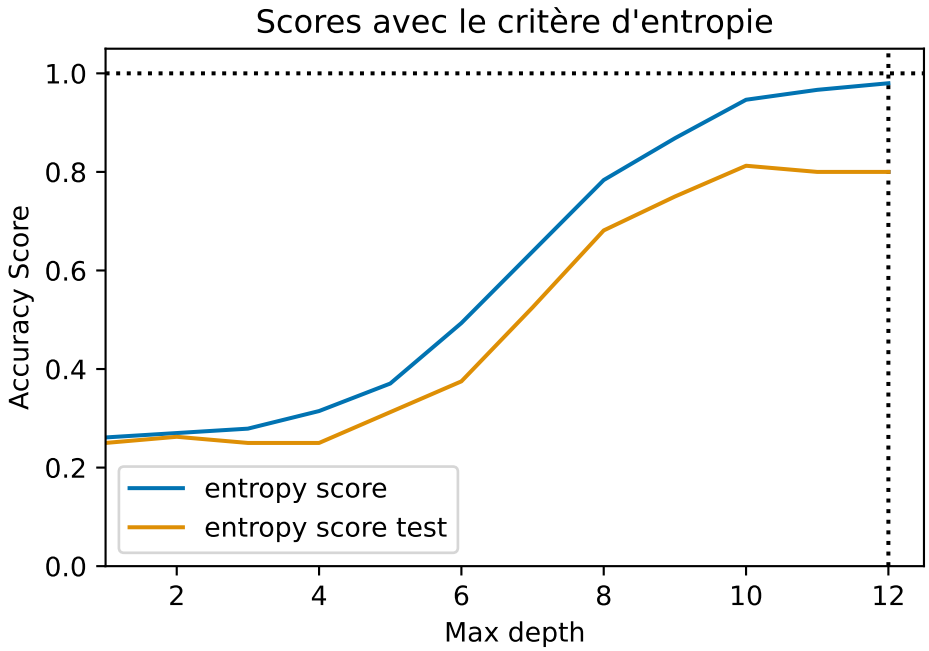
Q5. Créez n = 160 = 40 + 40 + 40 + 40 nouvelles données avec rand_checkers. Pour les arbres de décision entraînés précédemment, calculer la proportion d'erreurs faites sur cet échantillon de test. Commenter.

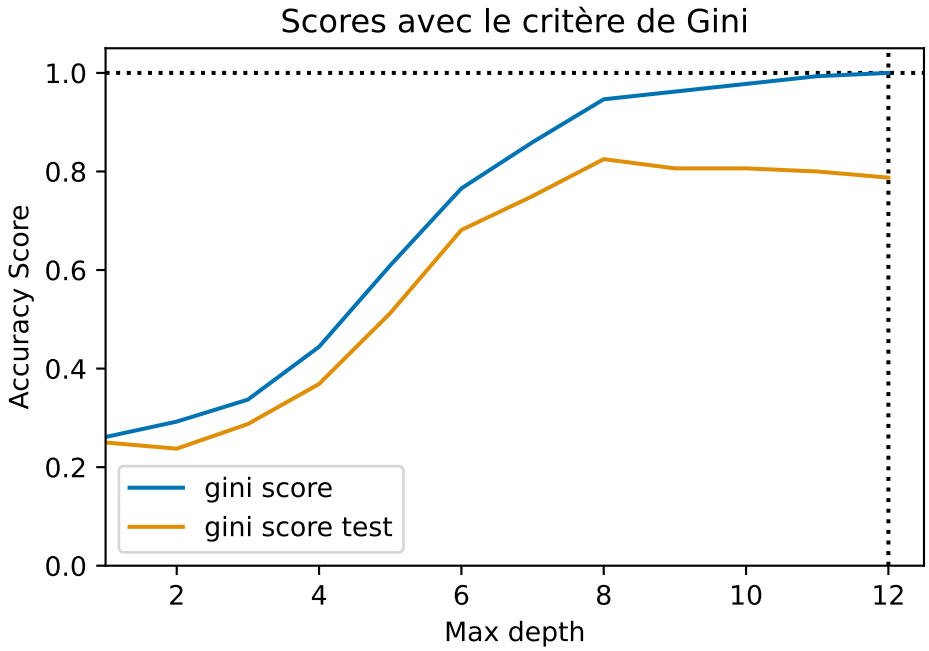
```
data_test = rand_checkers(40, 40, 40, 40) #On crée notre jeu de données de test.
X_test = data_test[:, :2]
Y_test = data_test[:, 2]

dmax = 12
scores_entropy_test = np.zeros(dmax)
scores_gini_test = np.zeros(dmax)

for i in range(dmax): # On calcule les scores pour les deux méthodes, pour le jeu de
                      # données test.
    dt_entropy = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = i+1)
    dt_entropy.fit(X, Y)
    scores_entropy_test[i] = dt_entropy.score(X_test, Y_test)

    dt_gini = tree.DecisionTreeClassifier(max_depth = i+1)
    dt_gini.fit(X, Y)
    scores_gini_test[i] = dt_gini.score(X_test, Y_test)

plt.figure() # On plot les scores de l'entropie pour le jeu de données de base et de test
             # selon la profondeur.
plt.xlim([1, 12.5])
plt.ylim([0, 1.05])
plt.axvline(x = 12, color = 'black', linestyle = "dotted")
plt.axhline(y = 1, color = 'black', linestyle = "dotted")
```

```
plt.plot(list(range(1, dmax + 1)), scores_entropy, label = "entropy score")
plt.plot(list(range(1, dmax + 1)), scores_entropy_test, label = "entropy score test")

plt.xlabel('Max depth')
plt.ylabel('Accuracy Score')
plt.title("Scores avec le critère d'entropie")
plt.legend()
plt.draw()

plt.figure() # On plot les scores du Gini pour le jeu de données de base et de test
             # selon la profondeur.
plt.xlim([1, 12.5])
plt.ylim([0, 1.05])
plt.axvline(x = 12, color = 'black', linestyle = "dotted")
plt.axhline(y = 1, color = 'black', linestyle = "dotted")

plt.plot(list(range(1, dmax + 1)), scores_gini, label = "gini score")
plt.plot(list(range(1, dmax + 1)), scores_gini_test, label = "gini score test")

plt.xlabel('Max depth')
plt.ylabel('Accuracy Score')
plt.title("Scores avec le critère de Gini")
plt.legend()

plt.draw()

print("Scores with entropy criterion: ", scores_entropy[:5])
print(scores_entropy[5:])
print("Scores with entropy criterion: ", scores_entropy_test[:5])
print(scores_entropy_test[5:])
print("Scores with Gini criterion: ", scores_gini[:5])
print(scores_gini[5:])
print("Scores with Gini criterion: ", scores_gini_test[:5])
print(scores_gini_test[5:])
```

```
Scores with entropy criterion:  [0.26116071 0.27008929 0.27901786 0.31473214 0.37053571]
[0.49330357 0.63839286 0.78348214 0.86830357 0.94642857 0.96651786
 0.97991071]
Scores with entropy criterion:  [0.25   0.2625 0.25   0.25   0.3125]
[0.375   0.525   0.68125 0.75    0.8125  0.8     0.8     ]
Scores with Gini criterion:  [0.26116071 0.29241071 0.33705357 0.44419643 0.609375  ]
[0.765625   0.859375   0.94642857 0.96205357 0.97767857 0.99330357
 1.        ]
Scores with Gini criterion:  [0.25    0.2375  0.2875  0.36875 0.5125 ]
```

```
[0.68125 0.75    0.825   0.80625 0.80625 0.8     0.7875 ]
```





On peut voir que les scores pour le jeu de données de test sont moins bons que pour le jeu de données de base. Cela peut être dû au fait que l'arbre de décision est trop complexe pour ce jeu de données.

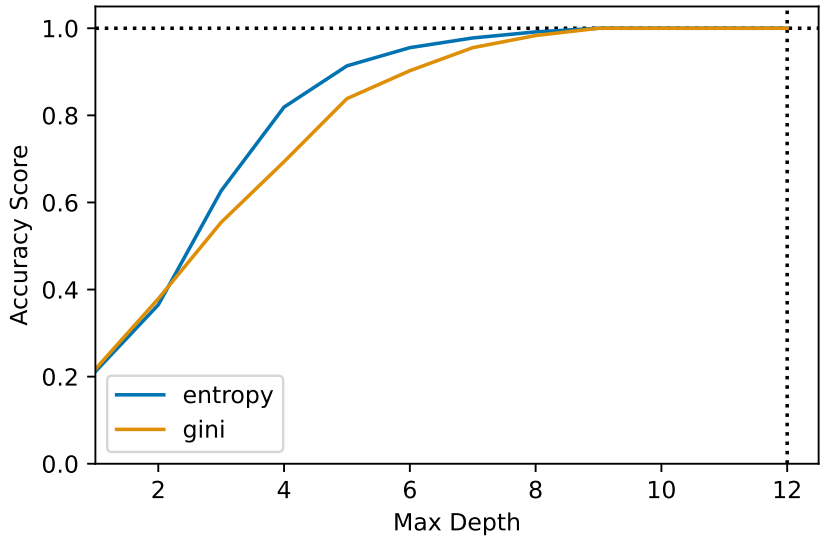
Q6. Reprendre les questions précédentes pour le dataset digits. Ce jeu de données est disponible dans le module sklearn.datasets. On peut l'importer avec la fonction load_digits du dit module

On fera une découpe en test/train de taille 80% - 20%.

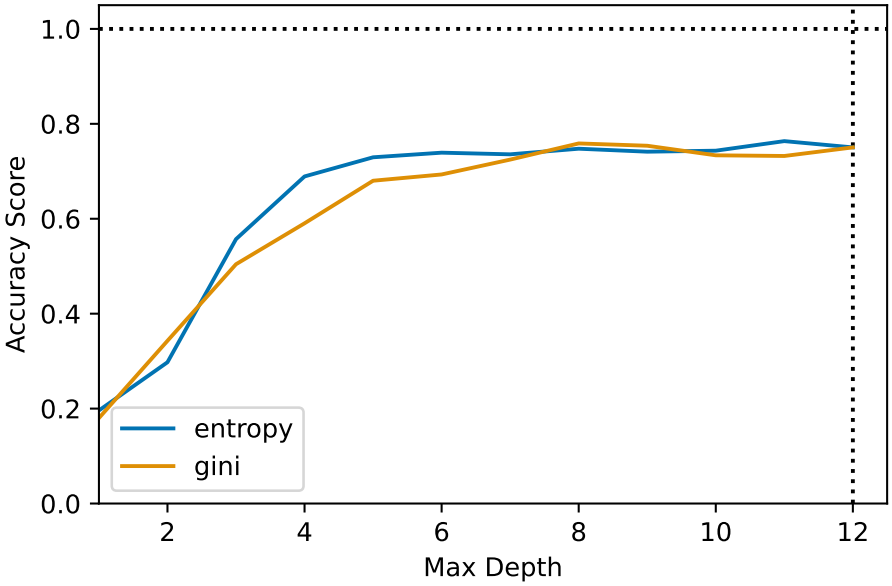
```
from sklearn.model_selection import train_test_split

digits = datasets.load_digits()
n_samples = len(digits.data)

X, X_test, Y, Y_test = train_test_split( # On sépare la base de données Digits,
                                         # avec un de size de 80%.
    digits.images.reshape(len(digits.data), -1), digits.target, test_size = 0.8)

dt_entropy = tree.DecisionTreeClassifier(criterion='entropy')
dt_gini = tree.DecisionTreeClassifier()

dt_entropy.fit(X, Y)
dt_gini.fit(X, Y)

dmax = 12
scores_entropy = np.zeros(dmax)
scores_gini = np.zeros(dmax)

for i in range(dmax): # On calcule les scores pour les deux méthodes,
                      # pour le jeu d'entrainements.
    dt_entropy = tree.DecisionTreeClassifier(criterion='entropy', max_depth=i+1)
    dt_entropy.fit(X, Y)
    scores_entropy[i] = dt_entropy.score(X, Y)

    dt_gini = tree.DecisionTreeClassifier(criterion='gini', max_depth=i+1)
    dt_gini.fit(X, Y)
    scores_gini[i] = dt_gini.score(X, Y)

plt.figure()

plt.xlim([1, 12.5])
plt.ylim([0, 1.05])
plt.axvline(x = 12, color = 'black', linestyle = "dotted")
plt.axhline(y = 1, color = 'black', linestyle = "dotted")

plt.plot(list(range(1, dmax + 1)), scores_entropy, label='entropy')
plt.plot(list(range(1, dmax + 1)), scores_gini, label='gini')
```

```
plt.xlabel('Max Depth')
plt.ylabel('Accuracy Score')
plt.title("Score selon la prondeur maximale de l'arbre de décision pour les données d'apprentis
plt.legend()

plt.draw()


scores_entropy = np.zeros(dmax)
scores_gini = np.zeros(dmax)

for i in range(dmax): # On calcule les scores pour les deux méthodes,
                      # pour le jeu de données test.
    dt_entropy = tree.DecisionTreeClassifier(criterion = "entropy", max_depth = i + 1)
    dt_entropy.fit(X, Y)
    scores_entropy[i] = dt_entropy.score(X_test, Y_test)

    dt_gini = tree.DecisionTreeClassifier(criterion = "gini", max_depth = i + 1)
    dt_gini.fit(X, Y)
    scores_gini[i] = dt_gini.score(X_test, Y_test)

plt.figure()

plt.xlim([1, 12.5])
plt.ylim([0, 1.05])
plt.axvline(x = 12, color = 'black', linestyle = "dotted")
plt.axhline(y = 1, color = 'black', linestyle = "dotted")

plt.plot(list(range(1, dmax + 1)), scores_entropy, label='entropy')
plt.plot(list(range(1, dmax + 1)), scores_gini, label='gini')

plt.xlabel('Max Depth')
plt.ylabel('Accuracy Score')
plt.title("Score selon la profondeur maximale de l'arbre de décision pour les données de test")
plt.legend()

plt.draw()
```

Score selon la prondeur maximale de l'arbre de décision pour les données d'apprentissage.



Score selon la profondeur maximale de l'arbre de décision pour les données de test



On voit rapidement que notre arbre de décision souffre porbablement d'overfitting. En effet, on a un score de 1 pour les données d'apprentissage, et un score de 0.7 pour les données de test. On peut donc supposer que notre arbre de décision est trop complexe pour ce jeu de données.

Méthodes de choix de paramètres - Sélection de Modèle

Q7. Utiliser la fonction sklearn.cross_validation.cross_val_score et tester la sur le jeu de données digits en faisant varier la profondeur de l'arbre de décision. On pourra se servir de cette fonction pour choisir la profondeur de l'arbre.

Estimer la meilleur profondeur avec un cross_val_score

```
from sklearn.model_selection import cross_val_score

dmax = 20

cv_scores = []

for i in range(dmax): # On calcule les scores pour l'entropie, pour le jeu de données
                      # d'apprentissage de Digits.
    tree_classifier = tree.DecisionTreeClassifier(criterion="entropy", max_depth = i + 1)
    scores = cross_val_score(tree_classifier, X, Y)
    mean_score = scores.mean()
    cv_scores.append(mean_score)

best_depth_entropy = cv_scores.index(max(cv_scores)) + 1 # On récupère la meilleure profondeur.

for i in range(dmax):
    print("Profondeur maximale =", i + 1,
    "Score de validation croisée moyen =", cv_scores[i])

print("Meilleure profondeur maximale (Entropy) =",
best_depth_entropy, ", Score de validation croisée moyen = {0:.5f}".format(max(cv_scores)))

print()

cv_scores = []

for i in range(dmax): # On calcule les scores pour le Gini, pour le jeu de données
                      # d'apprentissage de Digits.
    tree_classifier = tree.DecisionTreeClassifier(max_depth = i + 1)
    scores = cross_val_score(tree_classifier, X, Y)
    mean_score = scores.mean()
    cv_scores.append(mean_score)

best_depth_gini = cv_scores.index(max(cv_scores)) + 1 # On récupère la meilleure profondeur.

for i in range(dmax):
```

```
    print("Profondeur maximale =", i + 1,
    "Score de validation croisée moyen =", cv_scores[i])

print("Meilleure profondeur maximale (Gini) =",
best_depth_gini, ", Score de validation croisée moyen = {0:.5f}".format(max(cv_scores)))
```

```
Profondeur maximale = 1 Score de validation croisée moyen = 0.20618153364632236
Profondeur maximale = 2 Score de validation croisée moyen = 0.33712832550860716
Profondeur maximale = 3 Score de validation croisée moyen = 0.5237480438184664
Profondeur maximale = 4 Score de validation croisée moyen = 0.6825117370892018
Profondeur maximale = 5 Score de validation croisée moyen = 0.7215179968701095
Profondeur maximale = 6 Score de validation croisée moyen = 0.7214397496087637
Profondeur maximale = 7 Score de validation croisée moyen = 0.7356025039123631
Profondeur maximale = 8 Score de validation croisée moyen = 0.7466353677621282
Profondeur maximale = 9 Score de validation croisée moyen = 0.7382237871674491
Profondeur maximale = 10 Score de validation croisée moyen = 0.729773082942097
Profondeur maximale = 11 Score de validation croisée moyen = 0.7187793427230046
Profondeur maximale = 12 Score de validation croisée moyen = 0.7578247261345853
Profondeur maximale = 13 Score de validation croisée moyen = 0.7326682316118935
Profondeur maximale = 14 Score de validation croisée moyen = 0.7465571205007825
Profondeur maximale = 15 Score de validation croisée moyen = 0.7327856025039123
Profondeur maximale = 16 Score de validation croisée moyen = 0.7410406885758999
Profondeur maximale = 17 Score de validation croisée moyen = 0.738262910798122
Profondeur maximale = 18 Score de validation croisée moyen = 0.7215571205007825
Profondeur maximale = 19 Score de validation croisée moyen = 0.7438184663536775
Profondeur maximale = 20 Score de validation croisée moyen = 0.741001564945227
Meilleure profondeur maximale (Entropy) = 12 , Score de validation croisée moyen = 0.75782

Profondeur maximale = 1 Score de validation croisée moyen = 0.21725352112676055
Profondeur maximale = 2 Score de validation croisée moyen = 0.36201095461658844
Profondeur maximale = 3 Score de validation croisée moyen = 0.5067683881064162
Profondeur maximale = 4 Score de validation croisée moyen = 0.618114241001565
Profondeur maximale = 5 Score de validation croisée moyen = 0.6570813771517996
Profondeur maximale = 6 Score de validation croisée moyen = 0.7128716744913929
Profondeur maximale = 7 Score de validation croisée moyen = 0.7353677621283256
Profondeur maximale = 8 Score de validation croisée moyen = 0.72981220657277
Profondeur maximale = 9 Score de validation croisée moyen = 0.7493348982785604
Profondeur maximale = 10 Score de validation croisée moyen = 0.7604068857589984
Profondeur maximale = 11 Score de validation croisée moyen = 0.7688184663536776
Profondeur maximale = 12 Score de validation croisée moyen = 0.7466353677621284
Profondeur maximale = 13 Score de validation croisée moyen = 0.7604460093896713
Profondeur maximale = 14 Score de validation croisée moyen = 0.7381064162754304
Profondeur maximale = 15 Score de validation croisée moyen = 0.7465962441314554
Profondeur maximale = 16 Score de validation croisée moyen = 0.7464006259780908
```

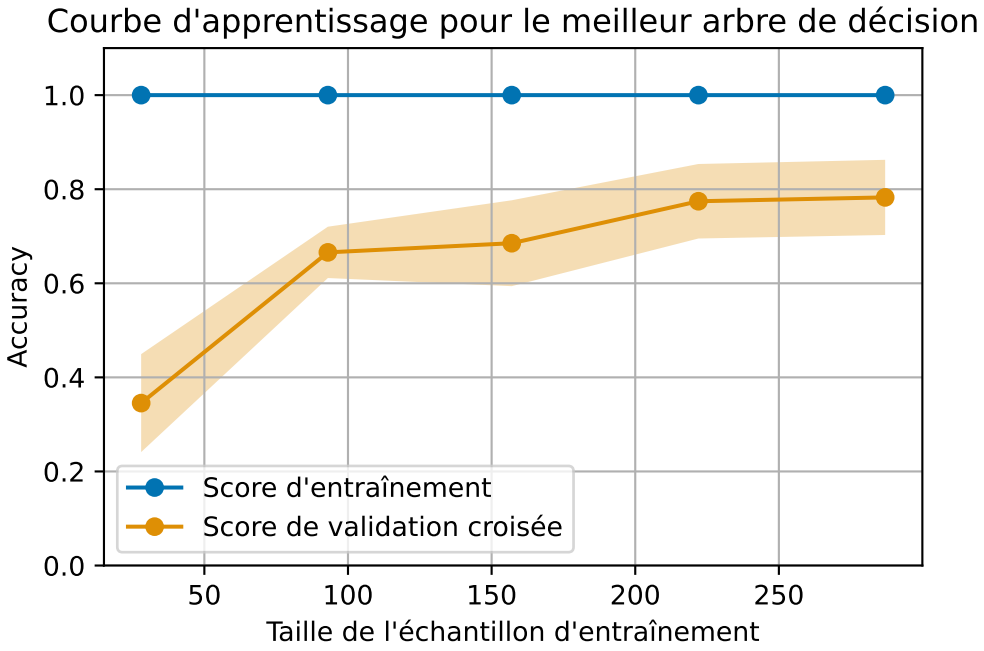
```
Profondeur maximale = 17 Score de validation croisée moyen = 0.7464397496087638
Profondeur maximale = 18 Score de validation croisée moyen = 0.7465179968701097
Profondeur maximale = 19 Score de validation croisée moyen = 0.7381846635367763
Profondeur maximale = 20 Score de validation croisée moyen = 0.7325508607198749
Meilleure profondeur maximale (Gini) = 11 , Score de validation croisée moyen = 0.76882
```

On peut voir qu'une profondeur trop élévée ne donne pas de meilleurs résultats, avec un coût en temps de calcul plus important.

Q8. En s'inspirant de http://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html afficher la courbe d'apprentissage (en : learning curve) pour les arbres de décisions sur le même jeu de données.

```
from sklearn.model_selection import learning_curve

X, X_test, Y, Y_test = train_test_split(
    digits.images.reshape(len(digits.data), -1), digits.target, test_size = 0.8)

n_samples, train_scores, test_scores = learning_curve(
    tree.DecisionTreeClassifier(criterion='entropy', max_depth = best_depth_entropy), X, Y)

train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure()
plt.grid()
plt.ylim([0, 1.1])

plt.fill_between(n_samples, train_scores_mean - 1.96 * train_scores_std,
                 train_scores_mean + 1.96 * train_scores_std, alpha = 0.3)
plt.fill_between(n_samples, test_scores_mean - 1.96 * test_scores_std,
                 test_scores_mean + 1.96 * test_scores_std, alpha=0.3)

plt.plot(n_samples, train_scores_mean, 'o-', label="Score d'entraînement")
plt.plot(n_samples, test_scores_mean, 'o-', label="Score de validation croisée")

plt.xlabel("Taille de l'échantillon d'entraînement")
plt.ylabel("Accuracy")
plt.title("Courbe d'apprentissage pour le meilleur arbre de décision")
plt.legend()
```

```
plt.draw()
```



On remarque que le score de validation croisée tends rapidement vers une limite et qu'une taille d'échantillon d'entrainement plus grande ne fournira pas de meilleurs résultats.