TP n°2 : Arbres

RENOIR Thamara

Commençons par importer tous les modules nécessaires à la réalisation de ce TP.

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

from sklearn import tree, datasets
from sklearn.model_selection import (train_test_split, cross_val_score,
LearningCurveDisplay, ShuffleSplit)
from tp_arbres_source import (rand_gauss, rand_bi_gauss, rand_tri_gauss,
                              rand_checkers, rand_clown,
                              plot_2d, frontiere)
```

1 Classification avec les arbres

1.1 Question 1

Dans le cadre de la régression une mesure d'homogénétié possible serait la variance de l'ensemble $R \in \mathbb{R}^p$:

$$\frac{1}{|R|} \sum_{i|x_i \in R} (y_i - \bar{y}_R)^2$$

où $\bar{y}_R = \frac{1}{|R|} \sum_{i|x_i \in R} y_i$.

En effet, cette mesure nous permettrait d'avoir des ensembles avec une variance minimale donc des ensembles homogènes.

1.2 Question 2

Simulons des échantillons de taille $n = 456$ avec la fonction `rand_checkers`.

```
np.random.seed(1)

n1 = 114
n2 = 114
n3 = 114
n4 = 114
data = rand_checkers(n1, n2, n3, n4)
```

Nos données sont réparties en quatre classes et nous les représentons graphiquement en Figure 1.

```
plt.ion()
plot_2d(data[:, :2], data[:, 2], w=None)
```

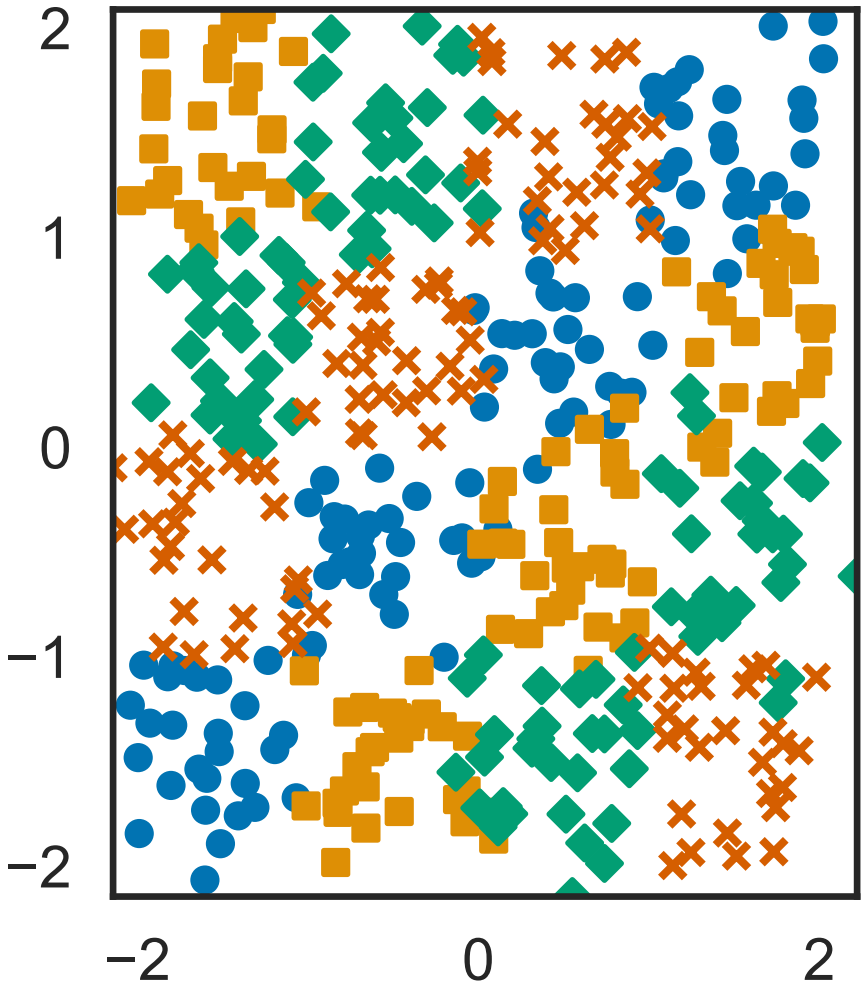

Figure 1: Jeu de données

Créons deux courbes (Figure 6), une pour l'indice Gini (en rouget) et une pour l'entropie (en vert), qui donnent le pourcentage d'erreurs commises en fonction de la profondeur maximale de l'arbre.

```
X_train = data[:, :2]
Y_train = data[:, 2].astype(int)

dmax = 12
error_entropy = np.zeros(dmax)
error_gini = np.zeros(dmax)
```

```
for i in range(dmax):
    dt_entropy = tree.DecisionTreeClassifier(criterion = "entropy",
    max_depth = i + 1)
    dt_entropy.fit(X_train, Y_train)
    error_entropy[i] = 1 - dt_entropy.score(X_train, Y_train)

    dt_gini = tree.DecisionTreeClassifier(criterion = "gini",
    max_depth = i + 1)
    dt_gini.fit(X_train, Y_train)
    error_gini[i] = 1 - dt_gini.score(X_train, Y_train)

plt.figure()
plt.plot(error_entropy * 100, 'g')
plt.plot(error_gini * 100, 'r')
plt.xlabel('Profondeur maximale')
plt.ylabel('Pourcentage d\'erreur')
plt.draw()
```

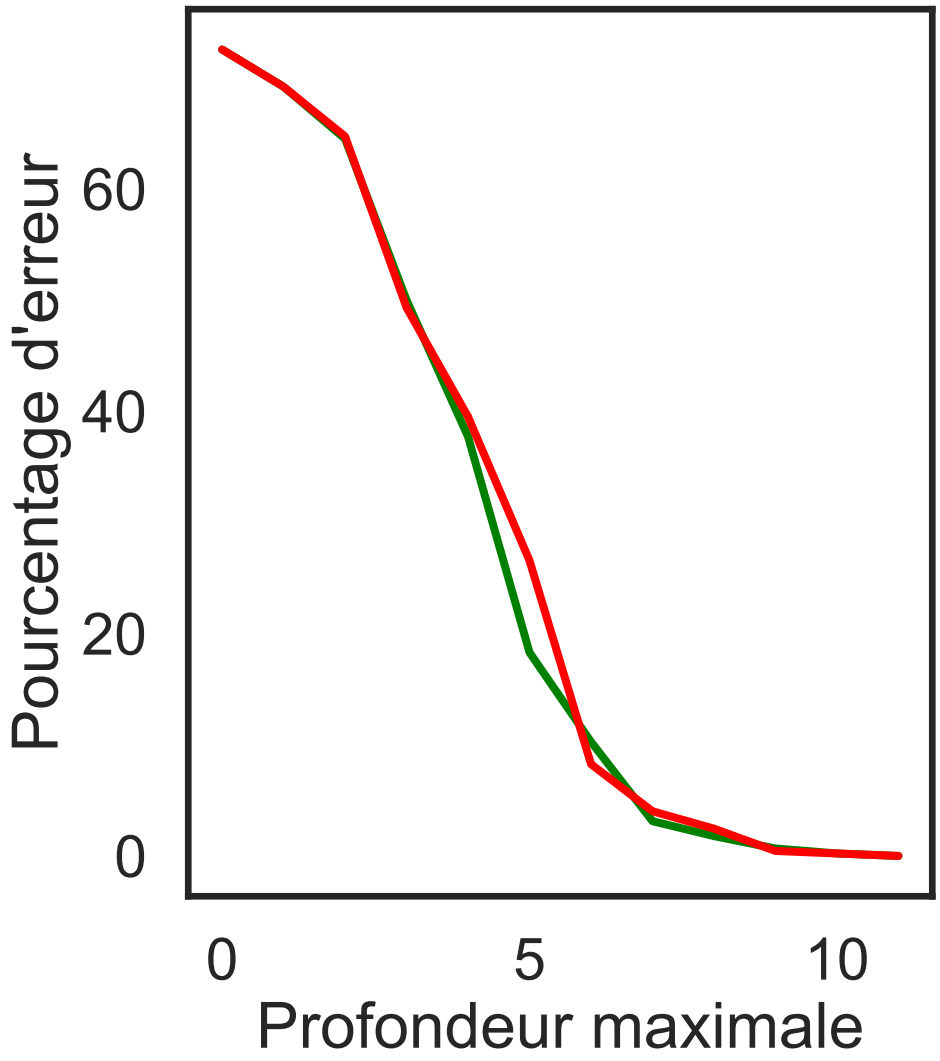

Figure 2: Pourcentage d'erreurs commises sur les données d'apprentissage

Nous constatons (Figure 2) que pour les deux critères, le pourcentage d'erreurs diminue quand la profondeur de l'arbre augmente, il tend vers 0. Cela ne signifie pas pour autant qu'un arbre avec un profondeur élevée permettra une meilleure prédiction.

Notons que nous calculons l'erreur à partir des données d'apprentissage. Nous sommes donc dans un cas de surapprentissage, où l'arbre construit est totalement adapté à la prédiction des données d'aprentissage, ce ne sera probablement pas le cas pour de nouvelles données.

Remarquons également que bien que les mesures d'impureté utilisées soient différentes, les pourcentages d'erreurs sont proches pour des pronfondeurs très petite sou très grandes. Dans

le premier cas le porucentage d'erreurs est nécessairement grand (environ 70% ici), tandis que dans le second il est proche de 0.

1.3 Question 3

Affichons la classification obtenue en utilisant la profondeur de l'arbre qui minimise le pourcentage d'erreurs obtenues avec l'entropie (Figure 3).

```
dt_entropy.max_depth = np.where(error_entropy == min(error_entropy))[0][0] + 1
dt_entropy.fit(X_train, Y_train)

plt.figure()
frontiere(lambda x: dt_entropy.predict(x.reshape((1, -1))), X_train,
Y_train, step=100)
plt.draw()
```

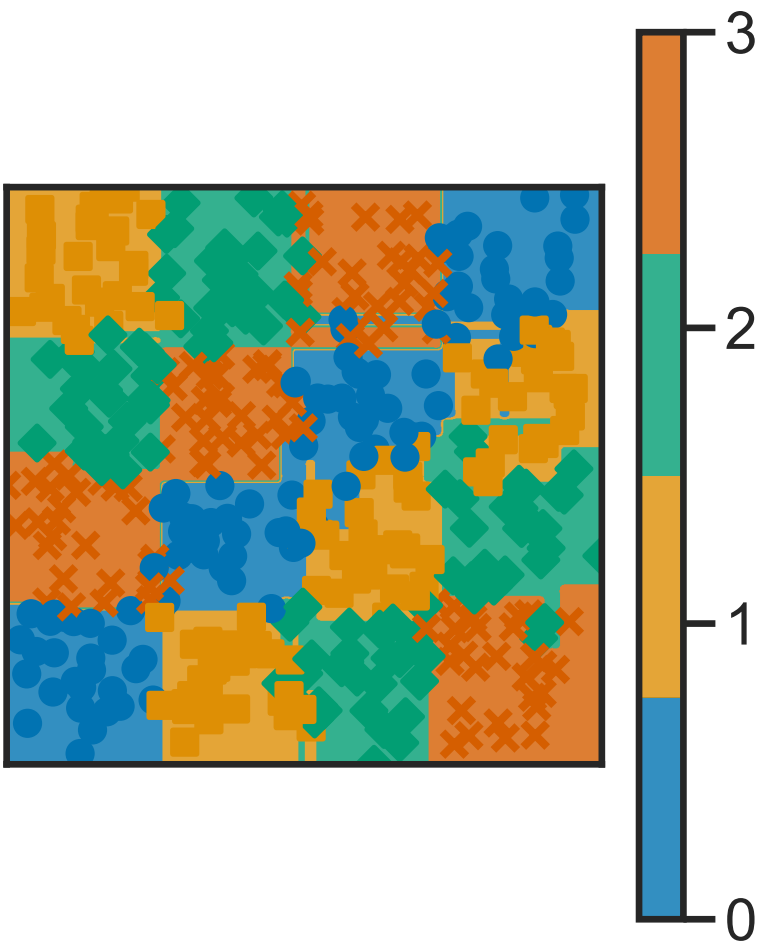

Figure 3: Meilleure classification des données avec le critère d'entropie

La majorité des données d'apprentissage sont assez bien classée. Cependant les frontières obtenus sont assez complexes puisqu'une profondeur maximale élevée signifie qu'on autorise une découpe assez fini de l'espace.

1.4 Question 4

Nous exportons un graphique de l'arbre obtenu à la question précédente au format pdf. Pour cela nous utilisons la fonction `export-graphiz`du module `tree`.

```
dot_data = tree.export_graphviz(dt_entropy, out_file=None)
graph = graphviz.Source(dot_data)
```

```
graph.render("Arbres\Arbre_checkers")
```

Nous obtenons une représentation de notre arbre de décision. Pour chaque nœud nous avons :

- son entropie
- le nombre d'observations
- le nombre d'observations par classe
- la variable et le seuil utilisés pour partitionner l'ensemble des observations du nœud

1.5 Question 5

Créons 160 nouvelles données avec `rand_checkers`. Elle nous servirons d'échantillon de test.

```
data_test = rand_checkers(40, 40, 40, 40)
X_test = data_test[:, :2]
Y_test = data_test[:, 2].astype(int)
```

Maintenant, nous reprenons les arbres de décision entraînés précédemment et nous calculons la proportion d'erreurs faites sur cet échantillon de test.

```
dmax = 12
error_entropy = np.zeros(dmax)
error_gini = np.zeros(dmax)

for i in range(dmax):
    dt_entropy = tree.DecisionTreeClassifier(criterion = "entropy",
    max_depth = i + 1)
    dt_entropy.fit(X_train, Y_train)
    error_entropy[i] = 1 - dt_entropy.score(X_test, Y_test)

    dt_gini = tree.DecisionTreeClassifier(criterion = "gini",
    max_depth = i + 1)
    dt_gini.fit(X_train, Y_train)
    error_gini[i] = 1 - dt_gini.score(X_test, Y_test)
```

Nous obtenons les courbes représentées en Figure 4.

La courbe verte est obtenue en utilisant l'entropie comme mesure d'impureté et la rouge en utilisant l'indicie Gini.

Comme précédemment le pourcentage d'erreurs diminu à mesure que la profondeur maximale de l'arbre augemente. Cependant, le minimum est supérieur de 10%, tandis que pour les

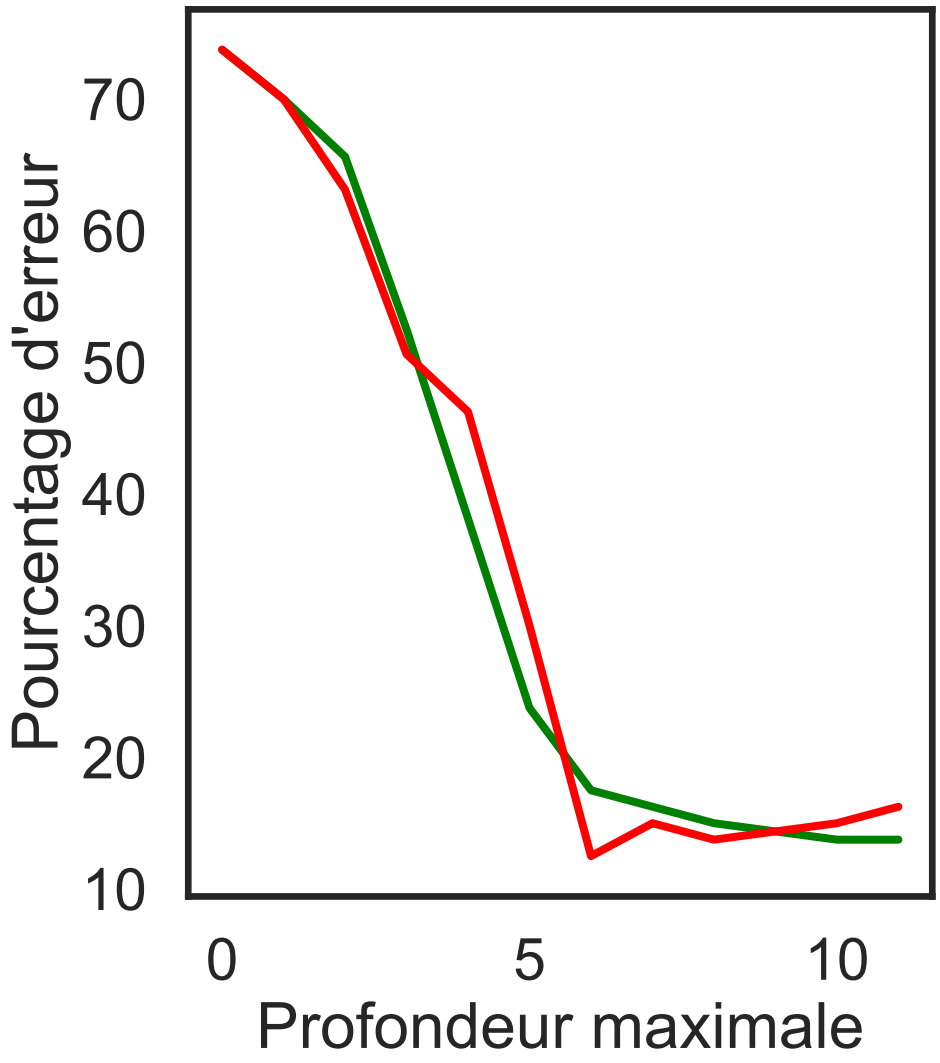

Figure 4: Pourcentage d'erreurs commises sur les données de test

données d'entraînement il était proche de 0%. Ce pourcentage semble même augmenter quand la profondeur maximale devient trop importante avec l'indice Gini.

Un arbre très profond, ne fournit pas nécessairement une meilleure prédiction.

1.6 Question 6

Nous allons maintenant utiliser les données du dataset DIGITS disponible dans le module `sklearn.datasets`.

```
digits = datasets.load_digits()
```

Il s'agit d'un ensemble d'images de 8x8 pixels. Nous en représentons 4 en Figure 5.

```
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, label in zip(axes, digits.images, digits.target):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title("Training: %i" % label)
```

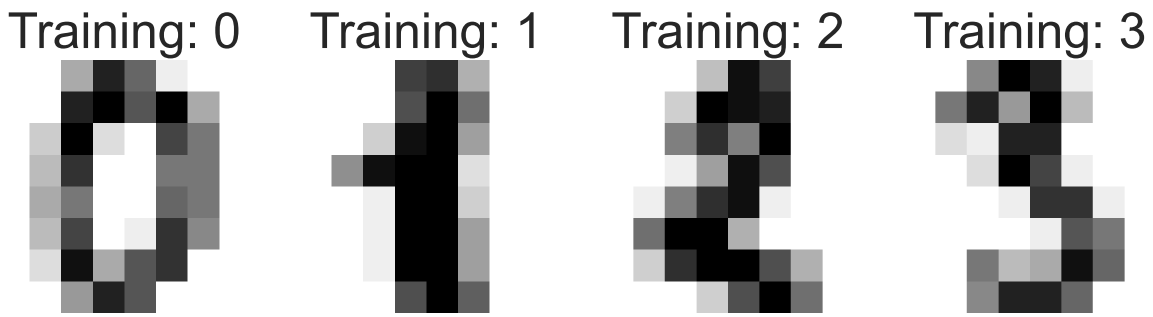

Figure 5: Exemple de données

Nous coupons notre échantillons en deux, 20% des données seront des données de test et les 80% restant seront les données d'apprentissage.

```
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

X_train, X_test, Y_train, Y_test = train_test_split(data,
digits.target, test_size = 0.2, shuffle=False)
```

Nous pouvons maintenant représenter les courbes qui donnent le pourcentage d'erreurs commises en fonction de la profondeur maximale de l'arbre pour les deux mesures d'impureté (l'indice Gini en rouge et l'entropie en vert). (Figure 6)

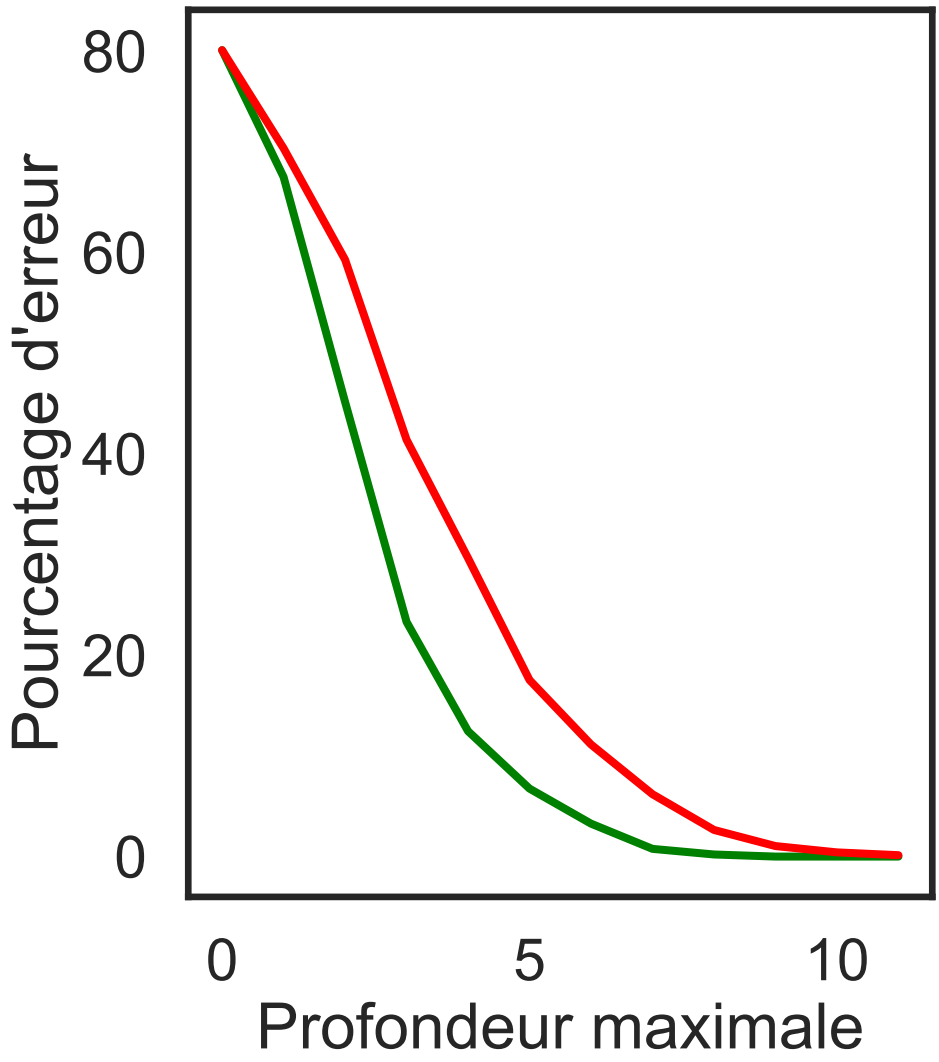

Figure 6: Pourcentage d'erreurs commises sur les données d'entraînement

Là encore, le pourcentage d'erreurs commises sur les données d'apprentissage décroit quand la profondeur de l'arbre croit. Il atteind 0% à partir d'une certaine profondeur dans les deux cas.

Nous visualisons en Figure 7 4 observations faisant partie de l'échantillon d'apprentissage et la prédiction obtenue avec l'arbre de décision qui minimise le pourcentage d'erreurs.

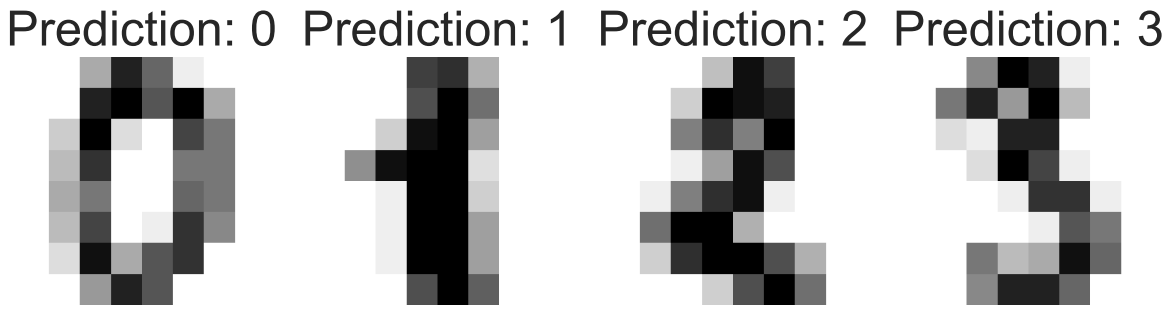

Figure 7: Exemple de prédiction des données d'apprentissage

Les 4 images sont correctement prédites, ce qui est normal puisqu'à la profondeur de l'arbre choisie, le pourcentage d'erreurs commises est nul.

Nous exportons comme précédemment un graphique de l'arbre obtenu.

```
dot_data = tree.export_graphviz(dt_entropy, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("Arbres\Arbre_digits")
```

Enfin nous calculons la proportion d'erreurs faites par les arbres de décisions entraînés précédemment sur l'échantillon de test (Figure 8).

Nous constatons qu'avec l'échantillon de test, le pourcentage d'erreurs décroit jusqu'à une certaine profondeur puis il semble à nouveau augmenter. Cette fois il ne semble pas tendre vers, il y a toujours des erreurs de prédiction.

2 Méthodes de choix de paramètres - Sélection de modèle

2.1 Question 7

Testons la fonction `sklearn.cross_validation.cross_val_score`sur le jeu de données DIGITS.

Pour cahque profondeur maximale choisie nous calculons le score du classificateur.

```
X = digits.data
Y = digits.target

dmax = 12
N = 5
```

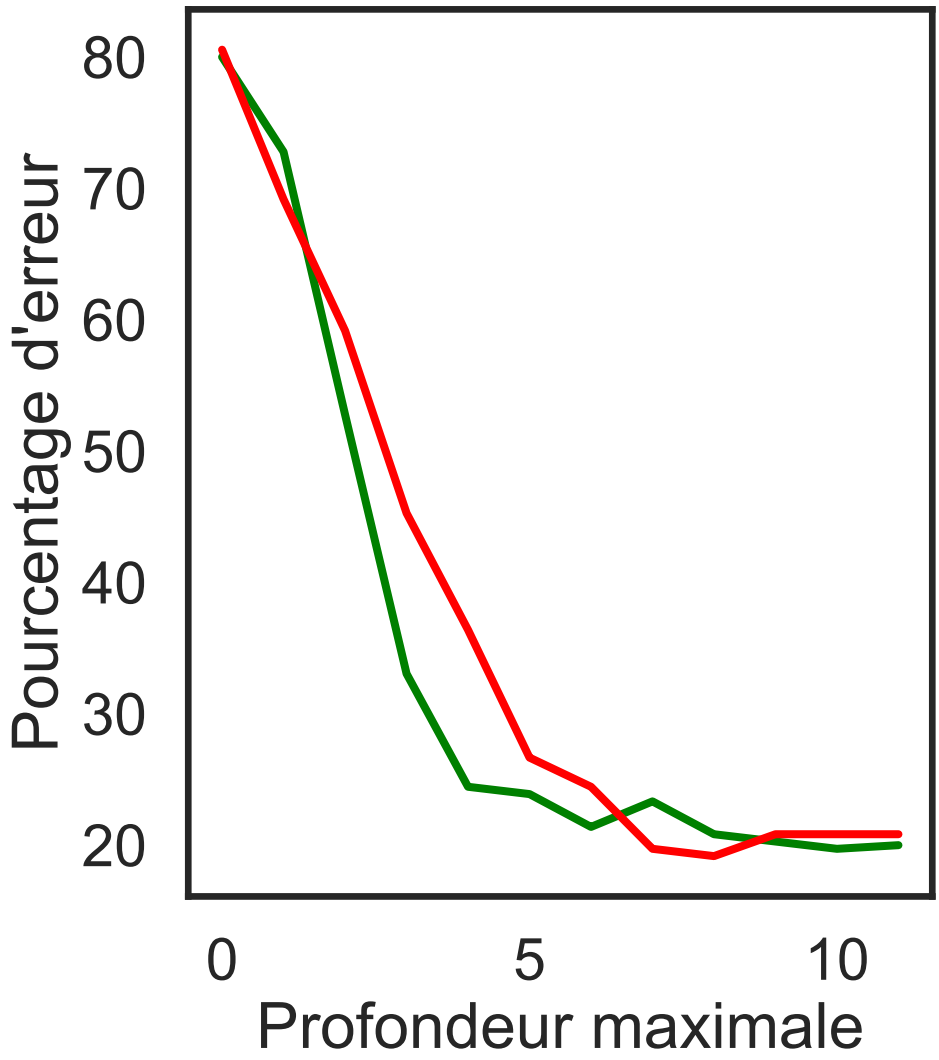

Figure 8: Pourcentage d'erreurs commises sur les données de test

```
score_entropy = np.zeros((dmax,N))
score_gini = np.zeros((dmax,N))

for i in range(dmax):
    dt_entropy = tree.DecisionTreeClassifier(criterion = "entropy",
    max_depth = i + 1)
    score_entropy[i] = cross_val_score(dt_entropy, X, Y, cv=N)

    dt_gini = tree.DecisionTreeClassifier(criterion = "gini",
    max_depth = i + 1)
    score_gini[i] = cross_val_score(dt_gini, X, Y, cv=5)
```

Nous obtenons les scores moyens représentés en Figure 9.

```
Moyennes des scores avec l'entropie :
 [0.195 0.334 0.503 0.632 0.735 0.774 0.801 0.798 0.818 0.806 0.806 0.82 ]
Moyennes des scores avec l'indice de Gini :
 [0.198 0.312 0.433 0.548 0.631 0.715 0.747 0.774 0.789 0.782 0.787 0.779]
```

Nous pouvons ensuite choisir la profondeur de l'arbre qui maximise la moyenne des scores.

```
Profondeur maximale avec l'entropie : 12
 Score : 0.819722995976478
Profondeur maximale avec l'indice de Gini : 9
 Score : 0.789134942742185
```

2.2 Question 8

Nous allons maintenant afficher la courbe d'apprentissage pour les arbres de décisions sur le même jeu de données en Figure 10.

```
dt_entropy = tree.DecisionTreeClassifier(criterion = "entropy",
max_depth = dmax_entropy)
dt_gini = tree.DecisionTreeClassifier(criterion = "gini",
max_depth = dmax_gini)

fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(10, 12), sharex=True)

common_params = {
    "X": X,
```

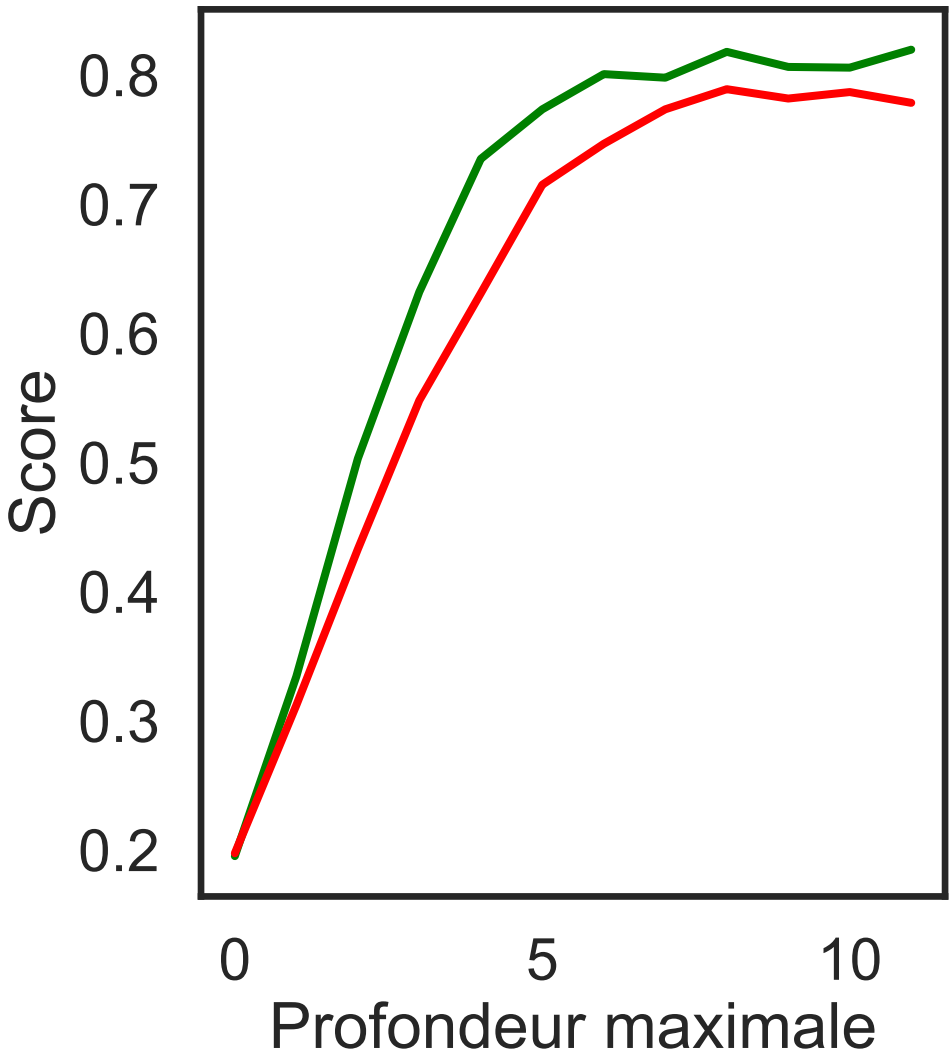

Figure 9: Moyenne des scores

```
    "y": Y,
    "train_sizes": np.linspace(0.1, 1.0, 5),
    "cv": ShuffleSplit(n_splits=50, test_size=0.2, random_state=0),
    "score_type": "both",
    "n_jobs": 4,
    "line_kw": {"marker": "o"},
    "std_display_style": "fill_between",
    "score_name": "Score",
}

for ax_idx, estimator in enumerate([dt_entropy, dt_gini]):
    LearningCurveDisplay.from_estimator(estimator, **common_params, ax=ax[ax_idx])
    handles, label = ax[ax_idx].get_legend_handles_labels()
    ax[ax_idx].legend(handles[:2], ["Training Score", "Test Score"])
    ax[ax_idx].set_title(f"Courbe d'apprentissage {estimator.criterion}")
```

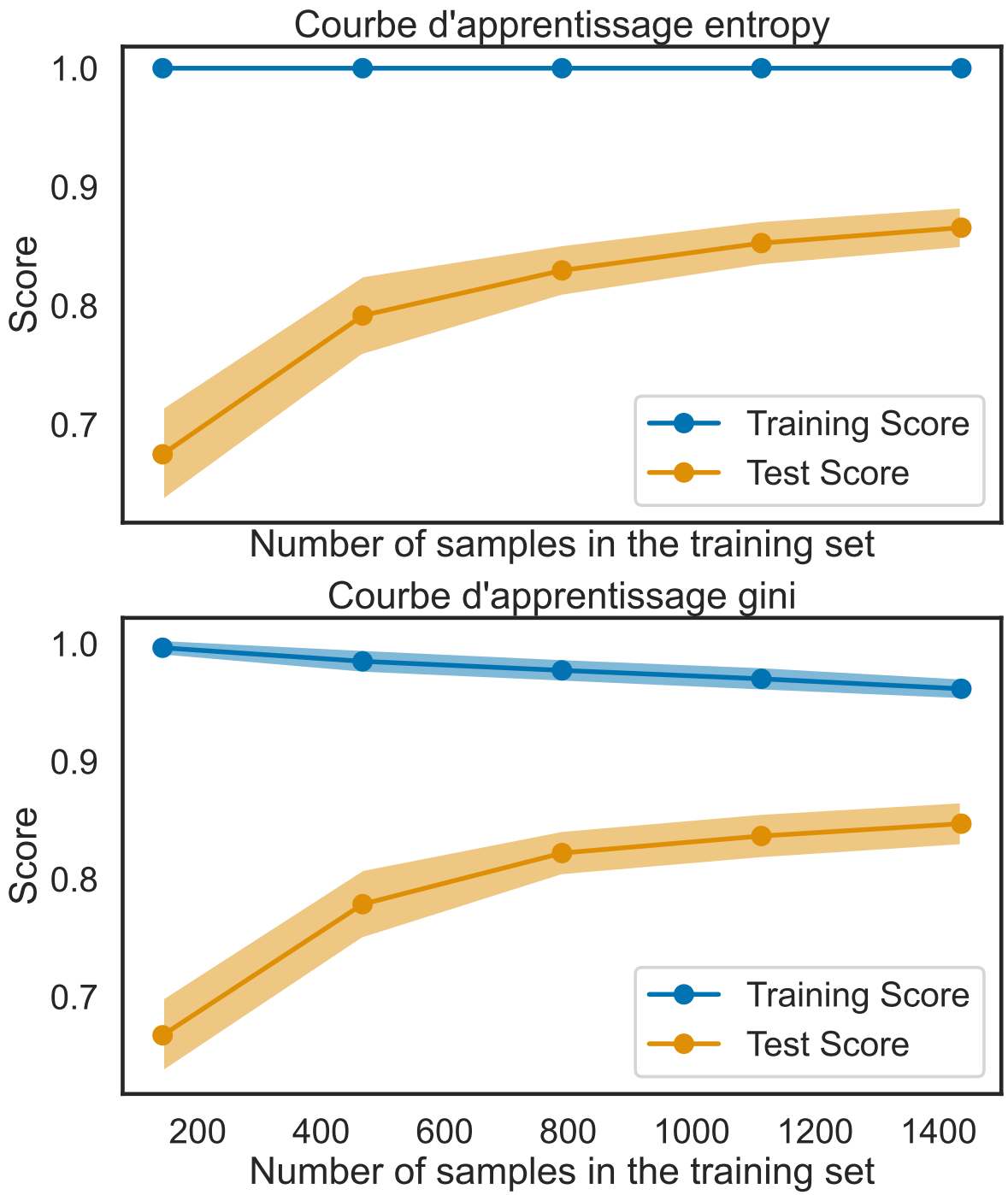

Figure 10: Courbes d'apprentissage pour le critère d'entropie et l'indice de Gini

Nous constatons que pour les deux critères le score de test reste très élevé, il est très proche ou égal à 1, cependant il décroit quand on utilise l'indice de Gini. A l'inverse les scores de test croient dans les deux cas.

Au vu de ces courbes, on peut supposer que l'ajout de données dans l'échantillon d'apprentissage nous permettrait d'améliorer les performences de notre modèle.