TP2 Arbres de décision

Dusfour-Castan Pauline

L'objectif de ce TP est de se pencher sur les arbres de décision à l'aide du module `tree` de scikit-learn, pour réaliser des classifications. Dans toute la suite de ce TP, nous nous sommes particulièrement aidé de la documentation de scikit-learn : https://scikit-learn.org/stable/user_guide.html

Cadre décisionnel et notations

- $\mathcal{Y}$ est l'ensemble des étiquettes données
- $x = (x_1, ..., x_p)^\mathcal{T} \in \mathcal{X}$ est une observation
- $\mathcal{D}_n = (x_i, y_i), i = 1, ..., n$ est l'ensemble d'apprentissage contenant les n observations et leurs étiquettes
- $\mathcal{K}$ est le nombre de classe

Tout d'abord on commence par installer les packages et fonctions (provenant du fichier `tp_arbres_source.py`) dont nous allons avoir besoin dans la suite 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.model_selection import cross_val_score
from sklearn import tree, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from tp_arbres_source import (rand_gauss, rand_bi_gauss, rand_tri_gauss,
                              rand_checkers, rand_clown,
                              plot_2d, frontiere)
```

Question 1

Dans le cadre de la régression, une autre mesure d'homogénéité que l'on pourrait utiliser serait par exemple le coefficient de détermination R^2. Ce coefficient permet de mesurer la qualité d'un modèle de régression linéaire, il est défini comme le rapport entre la variance expliquée par la régression sur la variance totale :

$$R^2 = \frac{\sum_i^n = 1(\hat{y}_i - \overline{y}^2)}{\sum_i^n = 1(y_i - \overline{y}^2)}$$

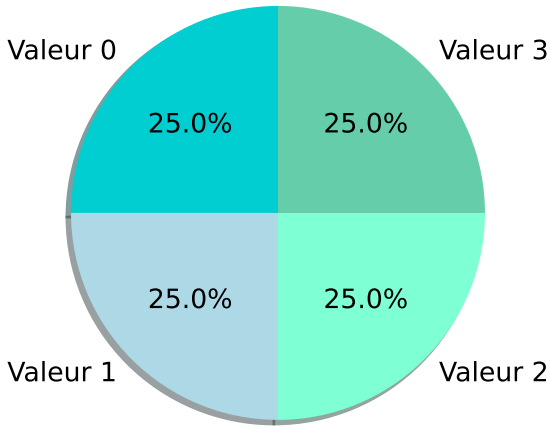
Question 2

Le but de cette question est de simuler, avec la fonction `rand_checkers`, un échantillon de taille n=456 et de construire nos premiers arbres.

```
np.random.seed(1998)
dt_entropy = tree.DecisionTreeClassifier(criterion="entropy")
# arbre de decision avec le critere de l'entropie
dt_gini = tree.DecisionTreeClassifier(criterion="gini")
# arbre de decision avec le critere indice de Gini

data = rand_checkers(n1=114, n2=114, n3=114, n4=114, sigma=0.1)
# creation d'echantillons
n_samples = len(data)
X = data[:,:2]
Y = data[:,2].astype(int)
```

Maintenant que nous avons créé notre échantillon, vérifions que les classes soient biens équilibrées :

```
sizes = []
for i in [0, 1, 2, 3]:
    sizes.append(np.sum(Y == i))

labels = ['Valeur 0', 'Valeur 1', 'Valeur 2',
          'Valeur 3']
plt.pie(sizes, labels=labels, colors=['darkturquoise', 'lightblue', 'aquamarine',
 'mediumaquamarine'], autopct='%1.1f%%',
        shadow=True, startangle=90)
plt.show()
```

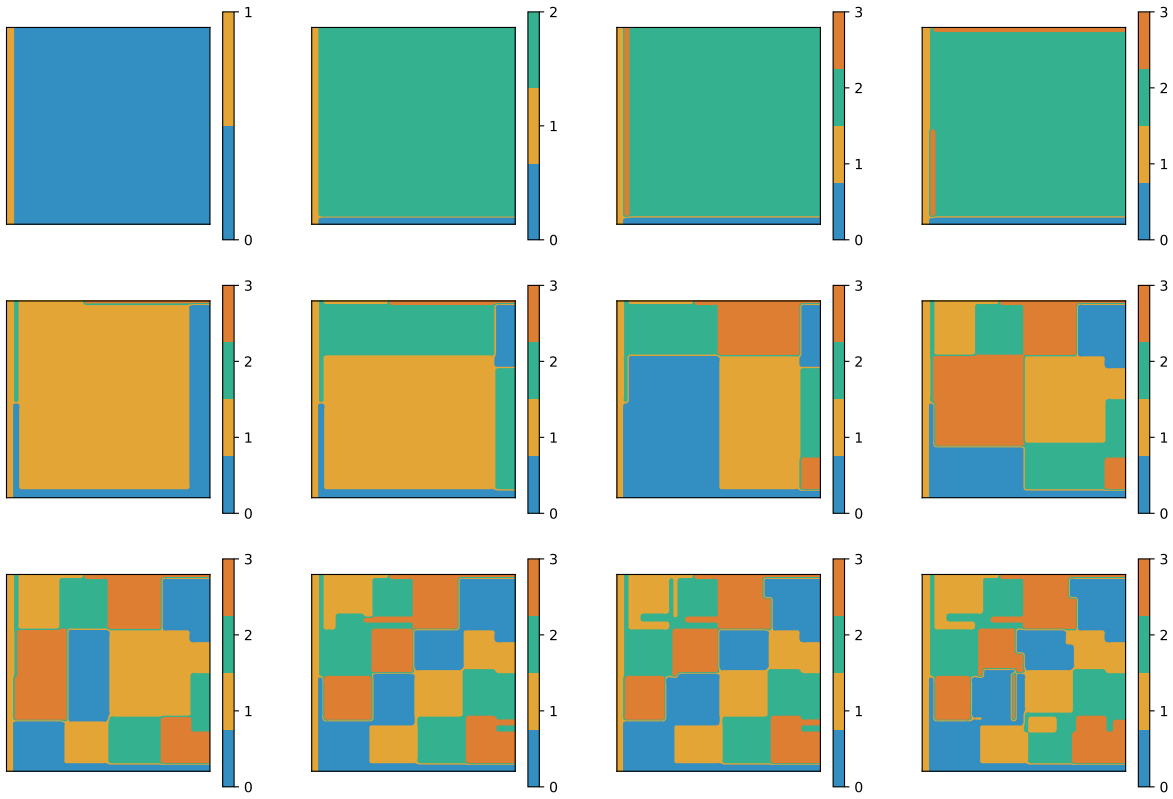



```
# On ajuste le modèle
dt_gini.fit(X, Y)
dt_entropy.fit(X, Y)

print("Gini criterion")
#print(dt_gini.get_params())
#print des parametres de l'arbre
print(dt_gini.score(X, Y))
# dt_gini.score --> calcule les performances du classifieur sur les
# données d'apprentissage

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

```
Gini criterion
1.0
Entropy criterion
1.0
```

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

```
plt.figure(figsize=(15, 10))
for i in range(dmax):
# on fait une boucle sur differentes valeurs de la profondeur de l'arbre
    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.subplot(3, 4, i + 1)
    frontiere(lambda x: dt_gini.predict(x.reshape((1, -1))), X, Y, step=50,
    samples=False)
plt.draw()
```



La figure ci-dessus nous donne les 12 premières classifications réalisées en fonction de la pro-

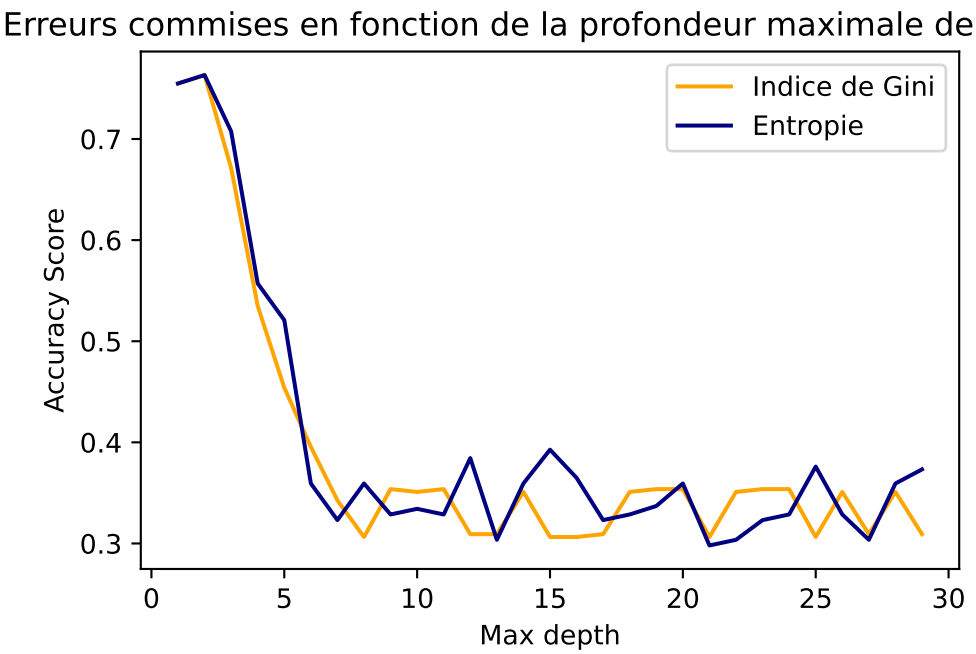
fondeur choisie.

Nous allons maintenant diviser notre échantillon en deux, un échantillon sera consacré à la partie apprentissage et un échantillon sera consacré à la partie test.

```
X_train1, X_test1, Y_train1, Y_test1 = train_test_split(X, Y, train_size=0.2)
```

```
score_gini1 = []
score_entropy1 = []
dmax1 = 30

for k in range(1, dmax1):
  dt_entropy = tree.DecisionTreeClassifier(
        criterion="entropy", max_depth=k)
  dt_entropy.fit(X_train1, Y_train1)
  predict_y_entropy = dt_entropy.predict(X_test1)
  score_entropy1.append(1-accuracy_score(predict_y_entropy, Y_test1))



  dt_gini = tree.DecisionTreeClassifier(
        criterion="gini", max_depth=k)
  dt_gini.fit(X_train1, Y_train1)
  predict_y_gini = dt_gini.predict(X_test1)
  score_gini1.append(1-accuracy_score(predict_y_gini, Y_test1))


plt.figure()
plt.plot(range(1, dmax1), score_gini1,
         color='orange', label='Indice de Gini')
plt.plot(range(1, dmax1), score_entropy1,
         color='navy', label='Entropie')
plt.xlabel("Max depth")
plt.ylabel("Accuracy Score")
plt.title("Erreurs commises en fonction de la profondeur maximale de l'arbre ")
plt.legend()
plt.draw()
print("Scores with entropy criterion: ", score_entropy1)
print("Scores with Gini criterion: ", score_gini1)
```

```
Scores with entropy criterion:  [0.754874651810585, 0.7632311977715878, 0.7075208913649025, 0
Scores with Gini criterion:  [0.754874651810585, 0.7632311977715878, 0.6713091922005572, 0.53
```



On peut remarquer que le pourcentage d'erreurs commises en fonction de la profondeur de l'arbre "stagne" (compris entre 3 et 4 %) à partir de 8, il n'est donc pas nécessaire de prendre une très grande profondeur d'arbre pour avoir une classification correcte. Prendre un grande profondeur d'arbre pourrait de toute façon nous être défavorable… on pourrait réaliser un sur-apprentissage qui ne nous apporterai rien de bon.

Question 3

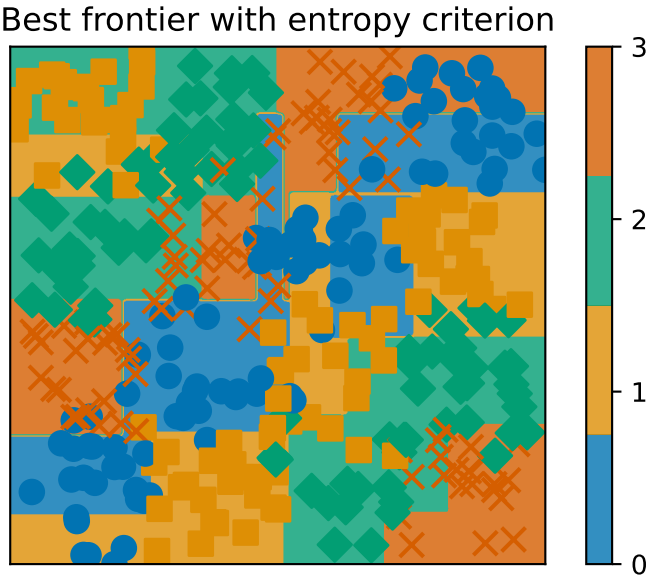
Nous allons maintenant afficher la classification obtenue en utilisant la profondeur de l'arbre qui minimise le pourcentage d'erreurs obtenues avec l'entropie.

```
profondeur_min = np.argmin(score_entropy1)
print("La profondeur qui minimise le pourcentage d'erreurs obtenues avec l entropie est")
print(profondeur_min, "pour un pourcentage d'erreur valant environ",
      np.round(score_entropy1[profondeur_min]*100, 3), "%")
```

```
La profondeur qui minimise le pourcentage d'erreurs obtenues avec l entropie est
20 pour un pourcentage d'erreur valant environ 29.805 %
```

```
dt_entropy.max_depth = profondeur_min

plt.figure()
frontiere(lambda x: dt_entropy.predict(x.reshape((1, -1))), X_test1, Y_test1, step=100)
# mange une fonction puis les donnees puis les labels
# trace les frontieres
# lambda : a x on asssocie l'evaluation en x de predict
# cree une grille tres fine et calcule pour chaque point la couleur et le trace
plt.title("Best frontier with entropy criterion")
plt.draw()
print("Best scores with entropy criterion: ", dt_entropy.score(X_test1, Y_test1))
```

```
Best scores with entropy criterion:  0.6267409470752089
```



Ci-dessus, nous pouvons observer la classification obtenue avec notre choix de profondeur maximale. Cette classification n'est pas parfaite, mais présente tout de même un bon partitionnement.

Question 4

Ci-dessous, vous pouvez visualiser l'arbre de décision obtenu :

```
dt_entropy_best = tree.DecisionTreeClassifier(criterion="entropy",
max_depth=profondeur_min+1)
dt_entropy_best2 = dt_entropy_best.fit(X_train1,Y_train1)
```

```
dot_data = tree.export_graphviz(dt_entropy_best2, out_file=None,
                                filled=True, rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph
graph.render("dt_arbre", directory="Figures")
```

```
'Figures\\dt_arbre.pdf'
```

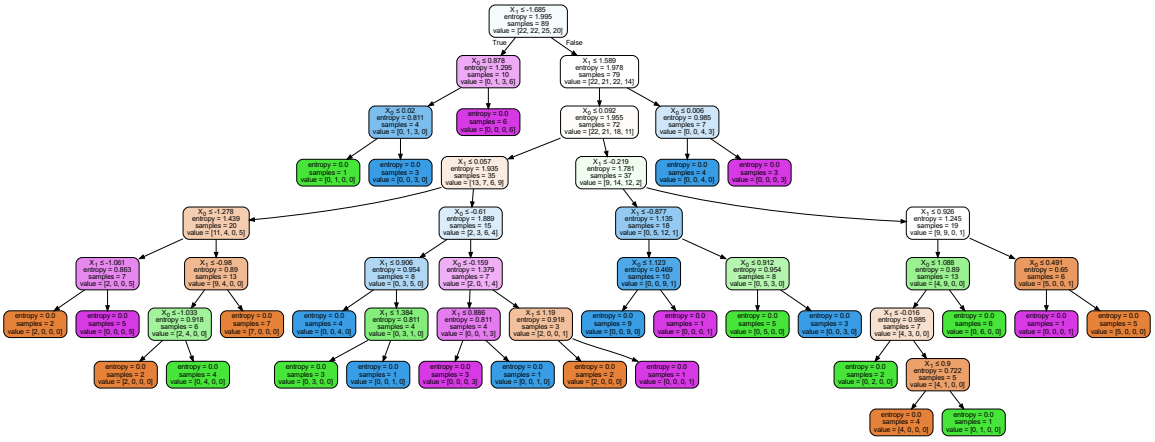


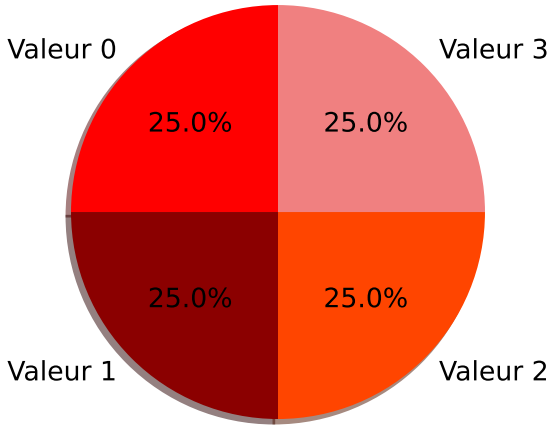
Figure 1: Arbre

Question 5

Cette question a pour objectif de tester les arbres de décision que nous avons entrainés précédemment. Pour cela, nous générons un échantillon de test à l'aide de la fonction `rand_checkers` que vous pouvez retrouver dans le fichier `tp_arbres_source.py` et on vérifie que nos classes soient bien équilibrées.

```
np.random.seed(1998)
data_test = rand_checkers(n1=40, n2=40, n3=40, n4=40, sigma=0.1)
X5 = data_test[:, :2]
```

```
Y5 = data_test[:, 2].astype(int)

# On vérifie que nos classes soient bien équilibrées
sizes2 = []
for i in [0, 1, 2, 3]:
    sizes2.append(np.sum(Y5 == i))

labels = ['Valeur 0', 'Valeur 1', 'Valeur 2',
          'Valeur 3']
plt.pie(sizes, labels=labels, colors=['red', 'darkred', 'orangered', 'lightcoral'],
 autopct='%1.1f%%',
        shadow=True, startangle=90)
plt.show()
```

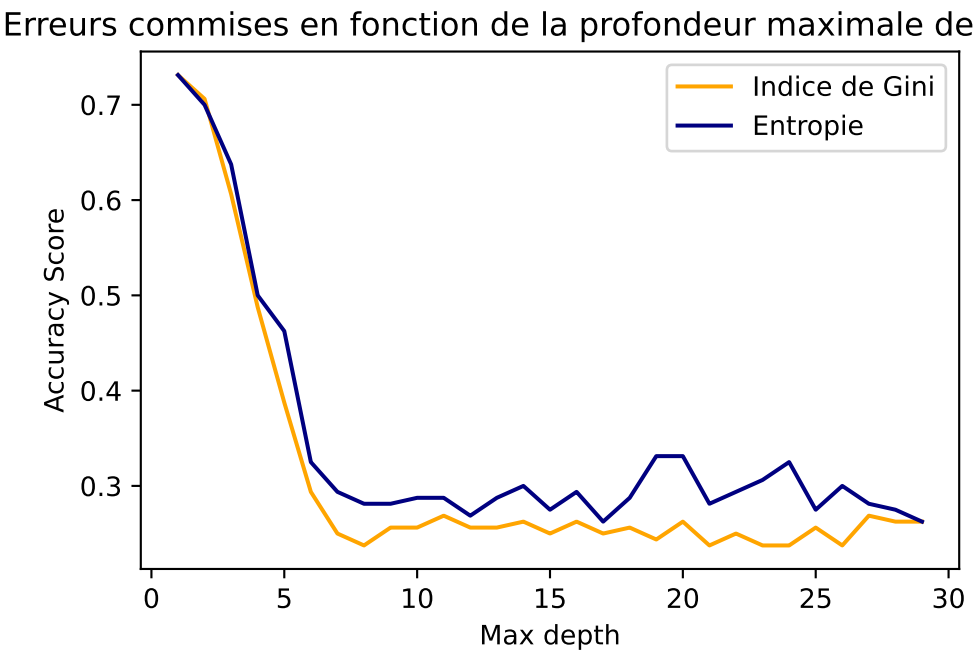



```
dmax = 30
scores_entropy2 = []
scores_gini2 = []

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

for i in range(1,dmax):
  dt_entropy = tree.DecisionTreeClassifier(
        criterion="entropy", max_depth=i)
  dt_entropy.fit(X_train1, Y_train1)
```

```
    predict_y_entropy5 = dt_entropy.predict(X5)
    scores_entropy2.append(1-accuracy_score(predict_y_entropy5, Y5))


    dt_gini = tree.DecisionTreeClassifier(
          criterion="gini", max_depth=i)
    dt_gini.fit(X_train1, Y_train1)
    predict_y_gini5 = dt_gini.predict(X5)
    scores_gini2.append(1-accuracy_score(predict_y_gini5, Y5))

plt.figure()
plt.plot(range(1, dmax), scores_gini2,
         color='orange', label='Indice de Gini')
plt.plot(range(1, dmax), scores_entropy2,
         color='navy', label='Entropie')
plt.xlabel("Max depth")
plt.ylabel("Accuracy Score")
plt.title("Erreurs commises en fonction de la profondeur maximale de l'arbre ")
plt.legend()
plt.draw()
print("Scores with entropy criterion: ", score_entropy1)
print("Scores with Gini criterion: ", score_gini1)
```

```
Scores with entropy criterion:  [0.754874651810585, 0.7632311977715878, 0.7075208913649025, 0
Scores with Gini criterion:  [0.754874651810585, 0.7632311977715878, 0.6713091922005572, 0.53
```

```
<Figure size 4500x3000 with 0 Axes>
```




```
profondeur_min5 = np.argmin(scores_entropy2)
print("La profondeur qui minimise le pourcentage d'erreurs obtenues avec l entropie est")
print(profondeur_min5, "pour un pourcentage d'erreur valant environ",
      np.round(scores_entropy2[profondeur_min5]*100, 3), "%")
```

```
La profondeur qui minimise le pourcentage d'erreurs obtenues avec l entropie est
16 pour un pourcentage d'erreur valant environ 26.25 %
```

On peut observer que le pourcentage d'erreurs commises décroit rapidemment pour de faibles valeurs de profondeur puis "stagne" à partir de 8 pour l'entropie.

Question 6

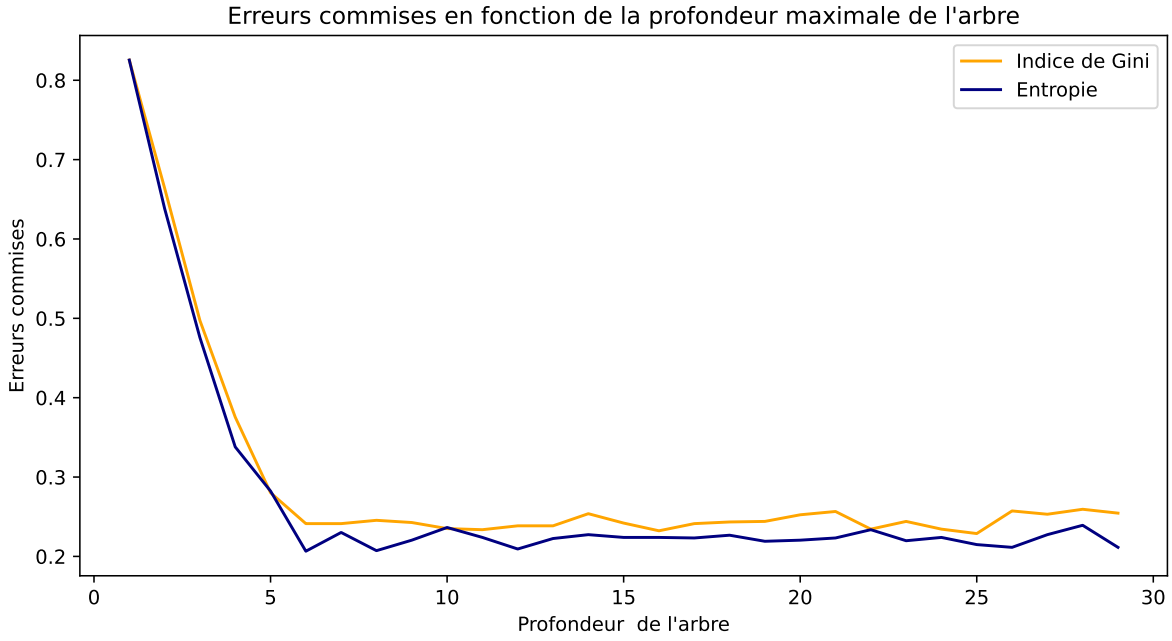
Nous allons maintenant reprendre les questions précédentes pour la dataset DIGITS disponible dans le sklearn.datasets.

A partir de ce datset, nous allons créer deux échantillons : * un échantillon de test (80 du jeu de données) * un échantillon d'entrainement (20 du jeu de données)

```
# On importe nos données et on crée nos différents échantillons

X_digits, Y_digits = datasets.load_digits(return_X_y=True)

X_digits_train2, X_digits_test2, Y_digits_train2, Y_digits_test2 = train_test_split(
    X_digits, Y_digits, train_size=0.2, random_state=123)
```

Ci-dessous les courbes donnant le pourcentage d'erreurs commises en fonction de la profondeur de l'arbre.

```
score_gini = []
score_entropy6 = []
pf_max = 30

for j in range(1, pf_max):

    dt_gini6 = tree.DecisionTreeClassifier(
        criterion="gini", max_depth=j)
    dt_gini6.fit(X_digits_train2, Y_digits_train2)
    predict_y_gini = dt_gini6.predict(X_digits_test2)
    score_gini.append(1-accuracy_score(predict_y_gini, Y_digits_test2))

    dt_entropy6 = tree.DecisionTreeClassifier(
        criterion="entropy", max_depth=j)
    dt_entropy6.fit(X_digits_train2, Y_digits_train2)
    predict_y_entropy = dt_entropy6.predict(X_digits_test2)
    score_entropy6.append(1-accuracy_score(predict_y_entropy, Y_digits_test2))



plt.figure(figsize=(10, 5))
plt.plot(range(1, pf_max), score_gini,
         color='orange', label='Indice de Gini')
plt.plot(range(1, pf_max), score_entropy6,
         color='navy', label='Entropie')
plt.xlabel("Profondeur  de l'arbre")
plt.ylabel("Erreurs commises")
plt.title("Erreurs commises en fonction de la profondeur maximale de l'arbre")
plt.legend(loc='upper right')
plt.show()
```



Comme pour toutes les autres courbes que l'on a vu jusqu'à présent, on peut observer une nette décroissance pour de faibles valeurs de profondeurs puis une stagnation à partir d'une certaine valeur.

Dans ce cas là, nous pourrions dire que notre pourcentage d'erreurs commises commence à stagner aux alentours d'une profondeur égale à 6.

```
La profondeur qui minimise le pourcentage d'erreurs obtenues avec l'entropie est
6 pour un pourcentage d'erreur valant 20.653685674547983 %
```

```
dot_data6 = tree.export_graphviz(dt_entropy_best6_fit, out_file=None,
                      filled=True, rounded=True,
                  special_characters=True)
graph6 = graphviz.Source(dot_data6)
graph6
graph6.render("digits_arbre", directory="Figures")
```

```
'Figures\\digits_arbre.pdf'
```

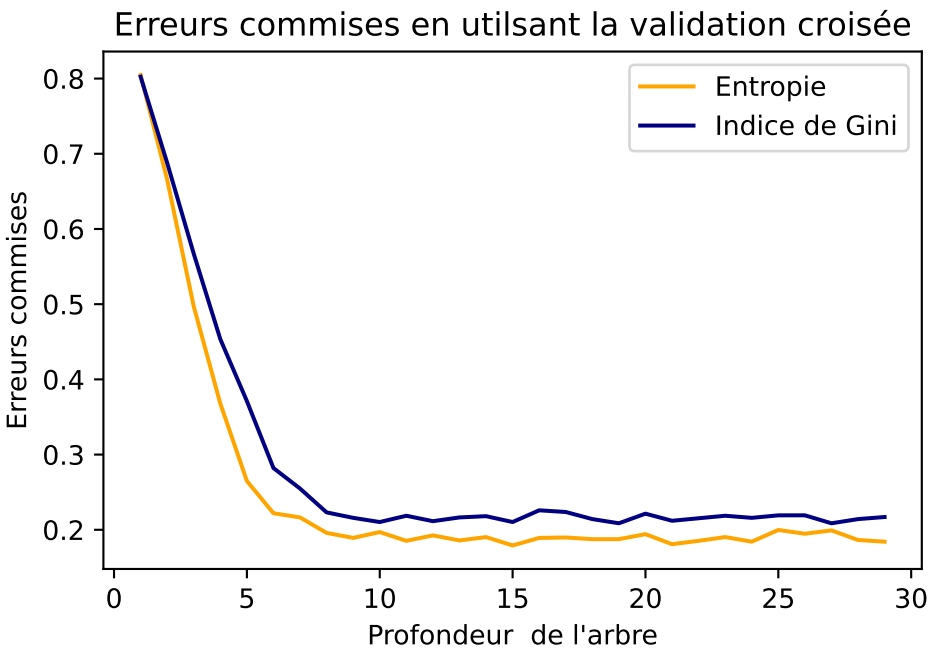
Nous n'avons malheureusement pas pu afficher l'arbre obtenu dans ce document à cause de problèmes de dimensions. Vous pouvez le retrouver sous le nom `digits_arbre.pdf` dans le dossier Figures.

Question 7

Dans cette question, nous allons réaliser de la validation croisée afin de sélectionner le meilleur modèle possible.

Pour cela, nous allons utiliser la fonction `sklearn.cross_validation.cross_val_score`. Nous la testerons sur le jeu de données DIGITS en faisant varier la profondeur de l'arbre de décision.

```
mean_score_entropy = []
mean_score_gini = []
score_entropy7 = []
score_gini7 = []
pf_max = 30

for pf in range(1, pf_max):
  dt_entropy7 = tree.DecisionTreeClassifier(criterion="entropy", max_depth=pf)
  score_entropy7 = cross_val_score(dt_entropy7, X_digits, Y_digits, cv=5, n_jobs=8)
  mean_score_entropy.append(1-score_entropy7.mean())

  dt_gini7 = tree.DecisionTreeClassifier(criterion="gini", max_depth=pf)
  score_gini7 = cross_val_score(dt_gini7, X_digits, Y_digits, cv=5, n_jobs=8)
  mean_score_gini.append(1-score_gini7.mean())

plt.plot(range(1, pf_max), mean_score_entropy,
         color='orange', label='Entropie')
plt.plot(range(1, pf_max), mean_score_gini,
         color='navy', label='Indice de Gini')
plt.xlabel("Profondeur  de l'arbre")
plt.ylabel("Erreurs commises")
plt.title("Erreurs commises en utilsant la validation croisée")
plt.legend(loc='upper right')
plt.show()
plt.figure(figsize=(10, 5))
```




```
<Figure size 3000x1500 with 0 Axes>

<Figure size 3000x1500 with 0 Axes>

La profondeur qui minimise le pourcentage d'erreurs obtenues avec l'entropie est
15 pour un pourcentage d'erreur valant 17.916 %
La profondeur qui minimise le pourcentage d'erreurs obtenues avec l'indice de Gini est
27 pour un pourcentage d'erreur valant 20.864 %
```

On obtient une profondeur de 11, selon le critère d'entropie, soit cinq "paliers" d'arbre en plus comparé à la profondeur trouvée lors de la question 6.

```
dt_entropy_best7 = tree.DecisionTreeClassifier(
    criterion="entropy", max_depth=pf_ent7 )
dt_entropy_best7.fit(X_digits_train2, Y_digits_train2)

print("L'erreur obtenue avec l'arbre entrainé et la profondeur minimisant le pourcentage")
print("d'erreurs vaut", np.round((1 - dt_entropy_best7.score(X_digits_test2,
Y_digits_test2))*100, 3), "%")
```

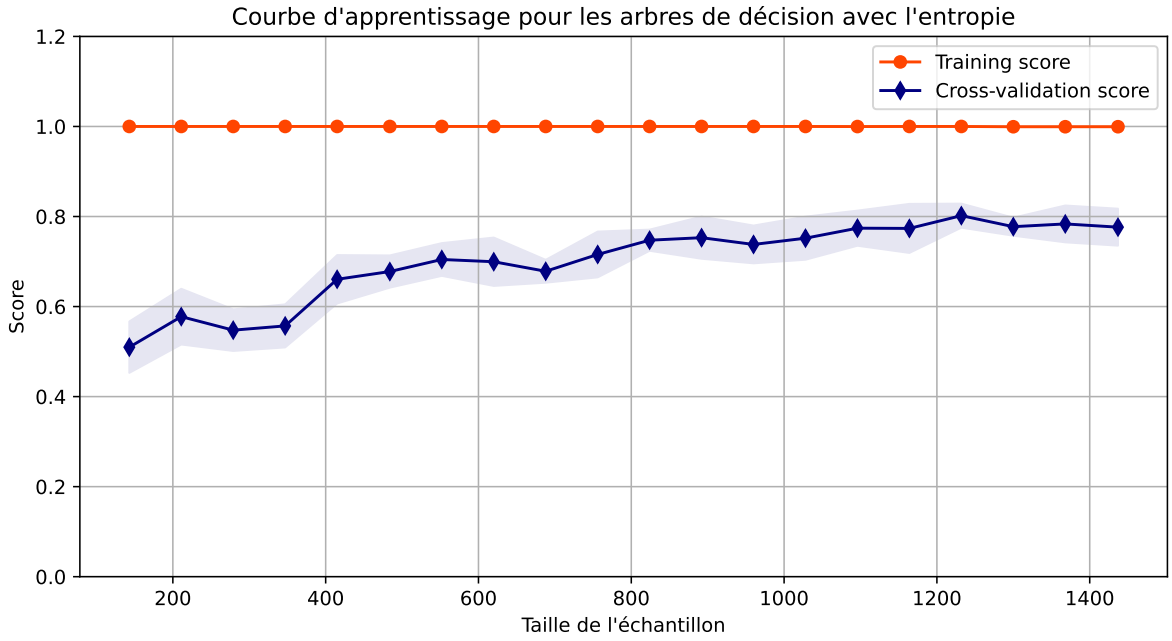
```
L'erreur obtenue avec l'arbre entrainé et la profondeur minimisant le pourcentage
d'erreurs vaut 22.184 %
```

Question 8

Dans cette dernière question, nous allons afficher la courbe d'apprentissage pour les arbres de décisions sur le même jeu de données. Nous nous sommes inspirer de https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html.

```
from sklearn.model_selection import learning_curve

_, axes = plt.subplots(1, 1, figsize=(10, 5))
axes.set_title(
    "Courbe d'apprentissage pour les arbres de décision avec l'entropie")
axes.set_xlabel("Taille de l'échantillon")
axes.set_ylabel("Score")

estimator_entropy = tree.DecisionTreeClassifier(
    max_depth=pf_ent7)
train_sizes, train_scores, test_scores, fit_times, _ = \
    learning_curve(estimator_entropy, X_digits, Y_digits, cv=5, n_jobs=8,
                   train_sizes=np.linspace(.1, 1.0, 20), return_times=True)

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)
fit_times_mean = np.mean(fit_times, axis=1)
fit_times_std = np.std(fit_times, axis=1)

axes.grid()
axes.fill_between(train_sizes, train_scores_mean - train_scores_std,
                  train_scores_mean + train_scores_std, alpha=0.1,
                  color="orangered")
axes.fill_between(train_sizes, test_scores_mean - test_scores_std,
                  test_scores_mean + test_scores_std, alpha=0.1,
                  color="navy")
axes.plot(train_sizes, train_scores_mean, 'o-', color="orangered",
          label="Training score")
axes.plot(train_sizes, test_scores_mean, 'd-', color="navy",
          label="Cross-validation score")
plt.ylim(0, 1.2)
axes.legend(loc="best")
```

```
<matplotlib.legend.Legend at 0x190529400a0>
```



Comme nous avons construit et entrainé notre modèle à partir des données d'apprentissage, on peut observer que (pour l'entropie), la courbe d'apprentissage est constante égale à 1 (donc le score est aussi égal à 1).

On peut également remarquer, que plus la taille de l'échantillon augmente, plus le score de cross-validation augmente et tend vers 0.8.

Ces deux informations nous laissent donc à penser que notre modèle est correct et que l'on ne réalise pas de sur-apprentissage.