Rapport TP2 Arbres

Mathieu Le Séac'h

Classification avec les arbres

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

from sklearn import tree, datasets, model_selection

from itertools import product
from functools import partial
from collections import defaultdict

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 d'une régression, on peut utiliser TODO comme mesure d'homogénéité.

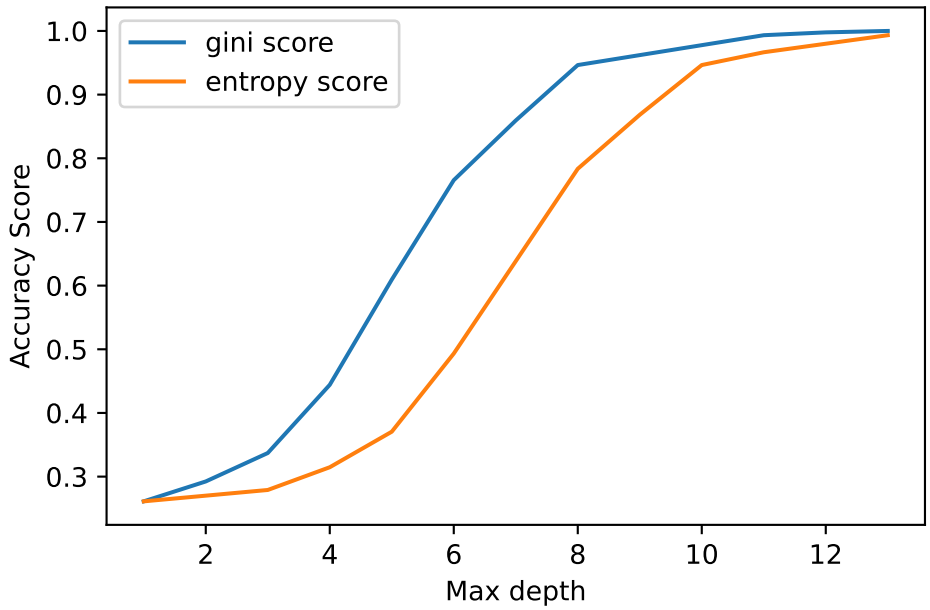
Q2)

On cherche à tracer les courbes donnant le pourcentage d'erreurs commises en fonction de la profondeur maximale de l'arbre. Pour ça on va entraîner nos arbres de décision sur des données générées puis on va tester la précision des prédictions sur ces mêmes données.

```
np.random.seed(10)

criterions = ["gini", "entropy"]
depths = range(1, 14)

n = 456
data = rand_checkers(n//4, n//4, n//4, n//4)
X_train = data[:, :2]
y_train = data[:, 2]

# (T -> S) -> Dict[K, List[T]] -> Dict[K, List[S]]
# applique une fonction à chaque valeur d'un dictionnaire de liste
def fmap_dict_list(f, dic):
    return { key: list(map(f, value)) for key, value in dic.items() }

# instancie tout les classificateurs
def create_all_classifiers(criterions, depths):
    classifiers = defaultdict(list)
    for (criterion, max_depth) in product(criterions, depths):
        classifier = tree.DecisionTreeClassifier(
            criterion=criterion,
            max_depth=max_depth,
        )

        classifiers[criterion].append(classifier)

    return classifiers

# entraîne tout les classificateurs sur les données X_train et y_train
def fit_all_classifiers(classifiers, X_train, y_train):
    return fmap_dict_list(
        lambda classifier: classifier.fit(X_train, y_train),
        classifiers
    )

# calcule tout les scores des classificateurs sur les données X_test et y_test
def compute_all_scores(classifiers, X_test, y_test):
    return fmap_dict_list(
        lambda classifier: classifier.score(X_test, y_test),
        classifiers
    )
```

```
classifiers = create_all_classifiers(criterions, depths)
classifiers = fit_all_classifiers(classifiers, X_train, y_train)
scores = compute_all_scores(classifiers, X_train, y_train)

print(scores)
plt.figure()
for criterion in scores.keys():
    plt.plot(depths, scores[criterion], label=f"{criterion} score")
    print(f"Scores with {criterion} criterion on training data: {scores[criterion]}")

plt.xlabel('Max depth')
plt.ylabel('Accuracy Score')
plt.legend()
plt.draw()
```

```
{'gini': [0.2611607142857143, 0.2924107142857143, 0.33705357142857145, 0.44419642857142855, 0
Scores with gini criterion on training data: [0.2611607142857143, 0.2924107142857143, 0.33705
Scores with entropy criterion on training data: [0.2611607142857143, 0.2700892857142857, 0.27
```



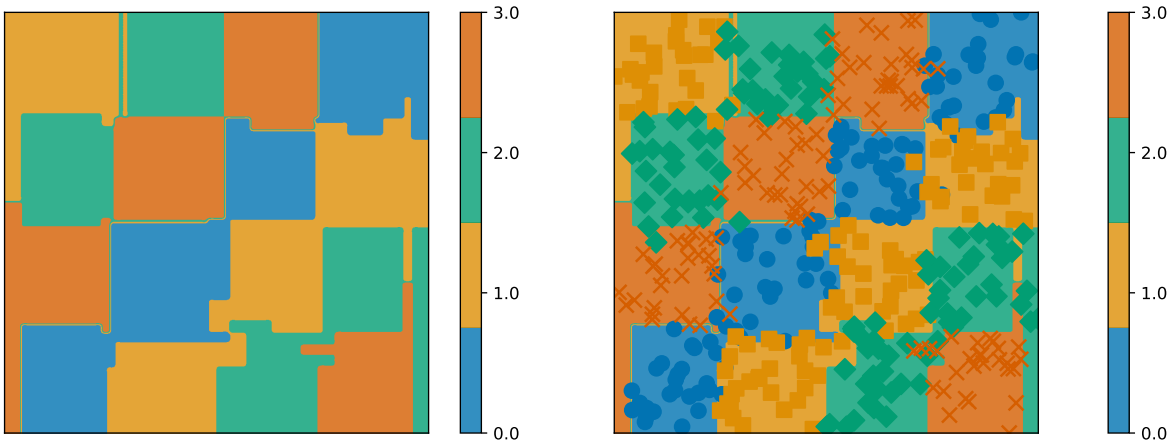
Comme on pouvait s'y attendre, la précision des arbres étant testés sur les données d'apprentissage, plus le modèle a de paramètres (ici de profondeur), mieux il va connaître les

données d'apprentissage et donc plus il va être précis dans ses prédictions. On atteint même une précision de 100% pour le critère de Gini et 99.3% pour le critère d'Entropie, ça semble indiquer soit que notre modèle est excellent, soit un cas de sur-apprentissage.

Q3)

Comme expliqué à la question précédente, plus la profondeur sera grande, plus l'erreur sera petite. Ainsi avec les profondeurs testées (de 1 à 13), l'arbre de décision qui minimise l'erreur est celui de profondeur 13, cependant dans notre cas le classificateur atteint déjà une précision parfaite à partir d'une profondeur de 16. Ainsi avec le critère d'entropie et une profondeur de 13 on obtient la classification suivante:

```
best_depth = np.argmax(scores["entropy"]) + 1
assert(best_depth == 13)

best_classifier = classifiers["entropy"][best_depth - 1]

plt.figure(figsize=(15, 10))
plt.subplot(2, 2, 1)
frontiere(
    lambda x: best_classifier.predict(x.reshape((1, -1))),
    X_train, y_train,
    step=100, samples=False
)

plt.subplot(2, 1, 1)
plot_2d(X_train, y_train)
frontiere(
    lambda x: best_classifier.predict(x.reshape((1, -1))),
    X_train, y_train,
    step=100, samples=False
)
```



Dans l'ensemble le damier ressemble bien à celui des données, cependant on remarque certains rectangles semblent assez exotiques et ne servent qu'à capturer une infîme partie des données. C'est un des symptômes du sur-apprentissage.

Q4)

Maintenant on cherche à visualiser les règles de décisions apprises par l'arbre dans un format intelligible. Pour ce faire on utilise la fonction `tree.export_graphviz` de scikit-learn et la librairie python `graphviz` qu'on exportera dans le fichier `graphviz/checkers.pdf`.

```
dot_data = tree.export_graphviz(
    best_classifier, out_file=None,
    feature_names=["x", "y"],
    filled=True, rounded=True,
    special_characters=True
)
graph = graphviz.Source(dot_data)
graph.render("graphviz/checkers")
```

```
'graphviz/checkers.pdf'
```

En accords avec le score de prédiction du classificateur, une immense majorité des feuilles de l'arbre de décision ont une entropie nulle, c'est à dire que le classificateur a trouvé une règle qui permet de prédire exactement la catégorie d'une donnée d'entraînement. De plus il y a aussi une grande partie de feuilles qui ne servent qu'à prédire une seule donnée d'entraînement (sample = 1), c'est à dire que le classificateur a créé une règle spécifique pour une seule donnée

observée. Comme pressenti à la question précédente, on est clairement dans un cas de sur-apprentissage.

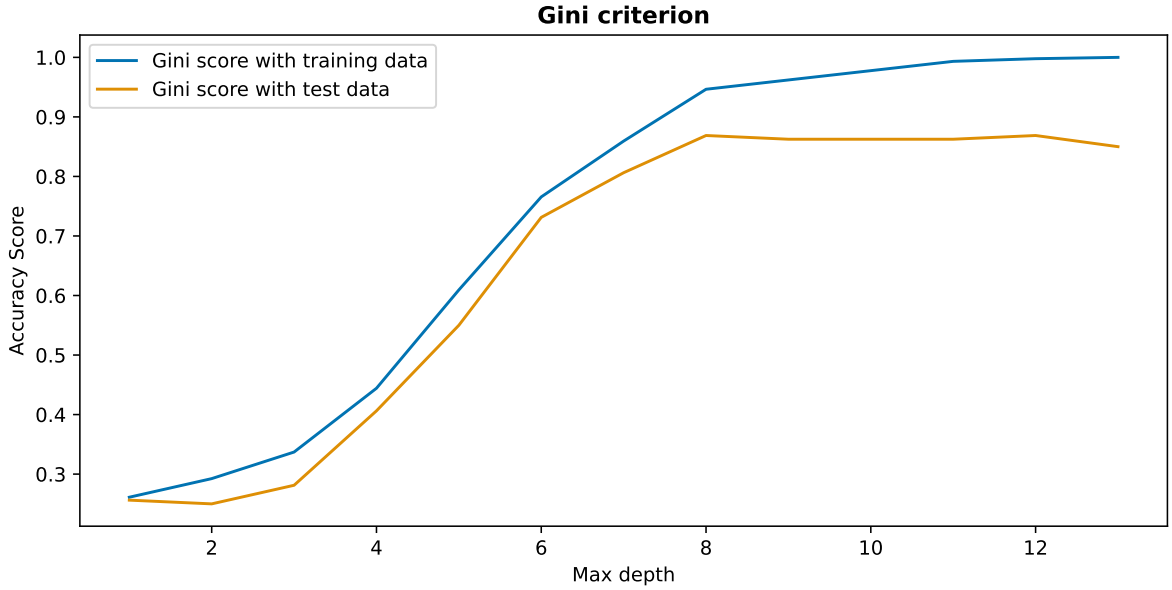
Q5)

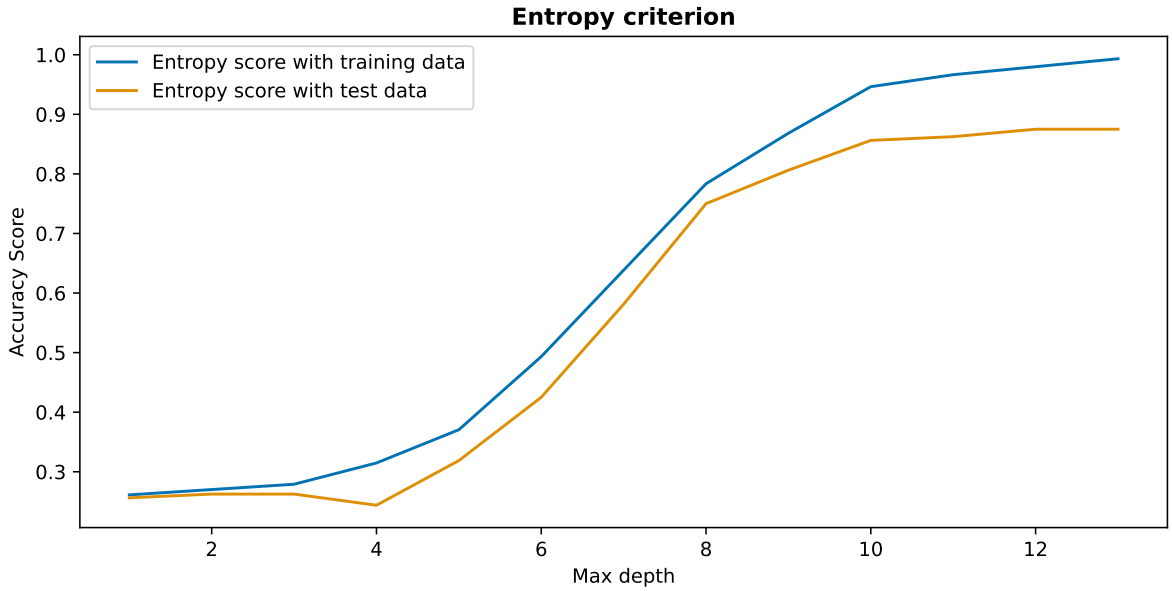
Pour avoir une estimation de la précision de notre arbre de décision plus précise, on va générer un échantillon de test et estimer la précision de notre classificateur sur ces données de test puis les comparer avec l'estimation de la précision sur les données d'entraînement.

```
n_test = 160
data_test = rand_checkers(n_test//4, n_test//4, n_test//4, n_test//4)
X_test = data_test[:, :2]
y_test = data_test[:, 2]

scores_test = compute_all_scores(classifiers, X_test, y_test)

plt.figure(figsize=(10, 10))
for i, criterion in enumerate(scores.keys()):
    plt.subplot(2, 1, i+1)
    plt.plot(
        depths, scores[criterion],
        label=f"{criterion.capitalize()} score with training data"
    )

    print(f"Scores with {criterion} criterion on training data: {scores[criterion]}")

    plt.plot(
        depths, scores_test[criterion],
        label=f"{criterion.capitalize()} score with test data"
    )

    print(f"Scores with {criterion} criterion on test data: {scores_test[criterion]}")

    plt.title(f"{criterion.capitalize()} criterion", weight="bold")
    plt.xlabel("Max depth")
    plt.ylabel("Accuracy Score")
    plt.legend()

plt.draw()
```

```
Scores with gini criterion on training data: [0.2611607142857143, 0.2924107142857143, 0.33705
```

```
Scores with gini criterion on test data: [0.25625, 0.25, 0.28125, 0.40625, 0.55, 0.73125, 0.8
Scores with entropy criterion on training data: [0.2611607142857143, 0.2700892857142857, 0.27
Scores with entropy criterion on test data: [0.25625, 0.2625, 0.2625, 0.24375, 0.31875, 0.425
```





Pour le critère de Gini on remarque que la précision sur les données de test cesse de croître à partir de la profondeur max de 8. De plus pour les deux critères (de Gini et d'entropie), on remarque que cette fois-ci la précision ne dépasse par le seuil des 90% et ce même en

augmentant la profondeur maximum de l'arbre. Donc pour chacun des critères l'estimation de la précision avec les données d'entraînement ont clairement surestimé la précision réelle de l'arbre de décision. Il n'y a plus de doute sur le fait qu'on est dans un cas de sur-apprentissage. En effet notre modèle prédit presque parfaitement les données d'entraînement mais il surestime sa capacité de généralisation à des données de test.

Q6)

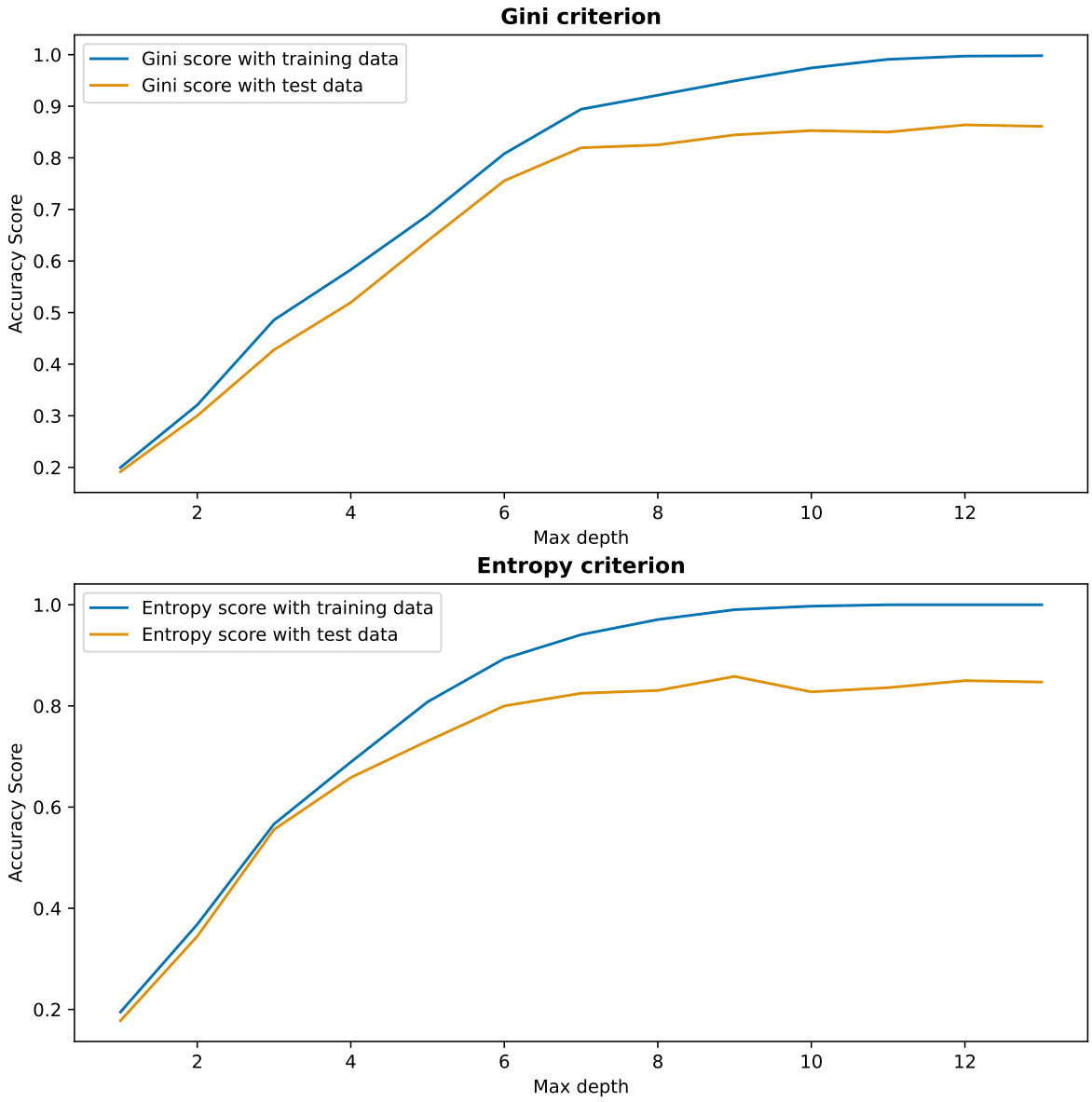
On cherche maintenant à reproduire les questions précédentes sur des données réelles issu du jeu de données ``digits'' fourni par la librairie scikit-learn.

```
X, y = datasets.load_digits(return_X_y=True)

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0.2, r

criterions = ["gini", "entropy"]
depths = range(1, 14)

classifiers = create_all_classifiers(criterions, depths)
classifiers = fit_all_classifiers(classifiers, X_train, y_train)

scores = compute_all_scores(classifiers, X_train, y_train)
scores_test = compute_all_scores(classifiers, X_test, y_test)

plt.figure(figsize=(10, 10))
for i, criterion in enumerate(scores.keys()):
    plt.subplot(2, 1, i+1)
    plt.plot(
        depths, scores[criterion],
        label=f"{criterion.capitalize()} score with training data"
    )

    print(f"Scores with {criterion} criterion on training data: {scores[criterion]}")

    plt.plot(
        depths, scores_test[criterion],
        label=f"{criterion.capitalize()} score with test data"
    )

    print(f"Scores with {criterion} criterion on test data: {scores_test[criterion]}")
```

```
    plt.title(f"{criterion.capitalize()} criterion", weight="bold")
    plt.xlabel("Max depth")
    plt.ylabel("Accuracy Score")
    plt.legend()

plt.draw()

dot_data = tree.export_graphviz(
    best_classifier, out_file=None,
    feature_names=["x", "y"],
    filled=True, rounded=True,
    special_characters=True
)
graph = graphviz.Source(dot_data)
graph.render("graphviz/digits")
```

```
Scores with gini criterion on training data: [0.19972164231036882, 0.32080723729993044, 0.485
Scores with gini criterion on test data: [0.19166666666666668, 0.3, 0.42777777777777776, 0.5
Scores with entropy criterion on training data: [0.19485038274182323, 0.36812804453723036, 0
Scores with entropy criterion on test data: [0.17777777777777778, 0.34444444444444444, 0.555
```

```
'graphviz/digits.pdf'
```



on retrouve exactement les mêmes problèmes de surapprentissage avec qu'avec nos données simulées. À savoir que sur les données d'entraînement on atteint une précision proche de 100% alors que sur les données de test on ne dépasse jamais le seuil de 90% de précision, donc une surestimation de la précision dans le premier cas. De plus en analysant l'arbre de décision généré (`graphviz/digits.pdf`), on observe le même phénomène avec la plupart des feuilles d'entropie nulle et avec seulement une ou deux donnée dedans.

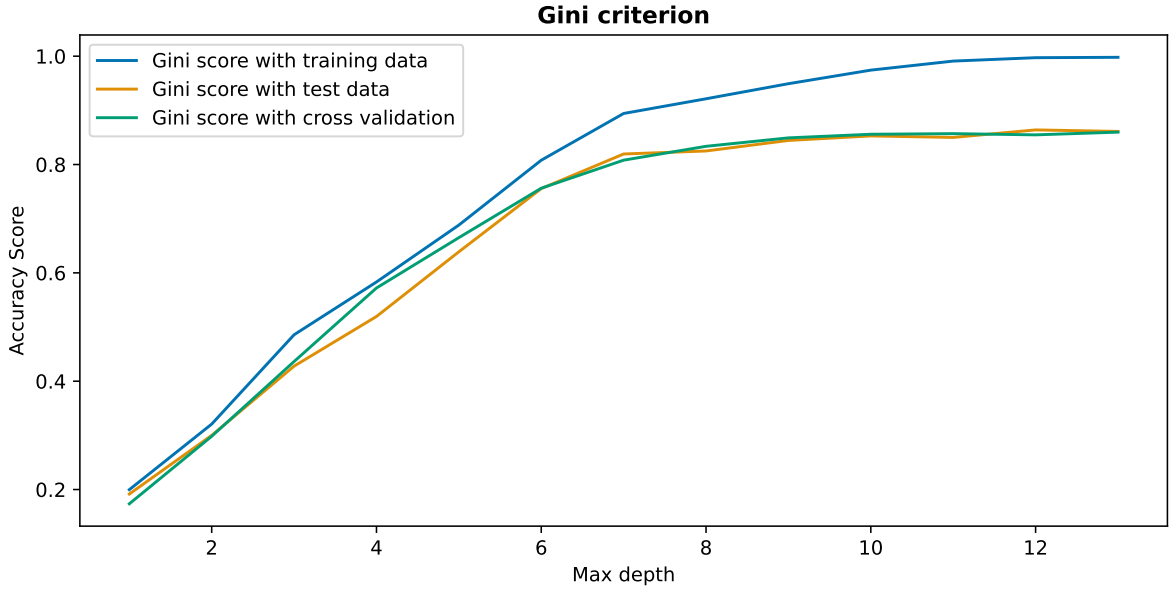
Méthodes de choix de paramètres - Sélection de modèle

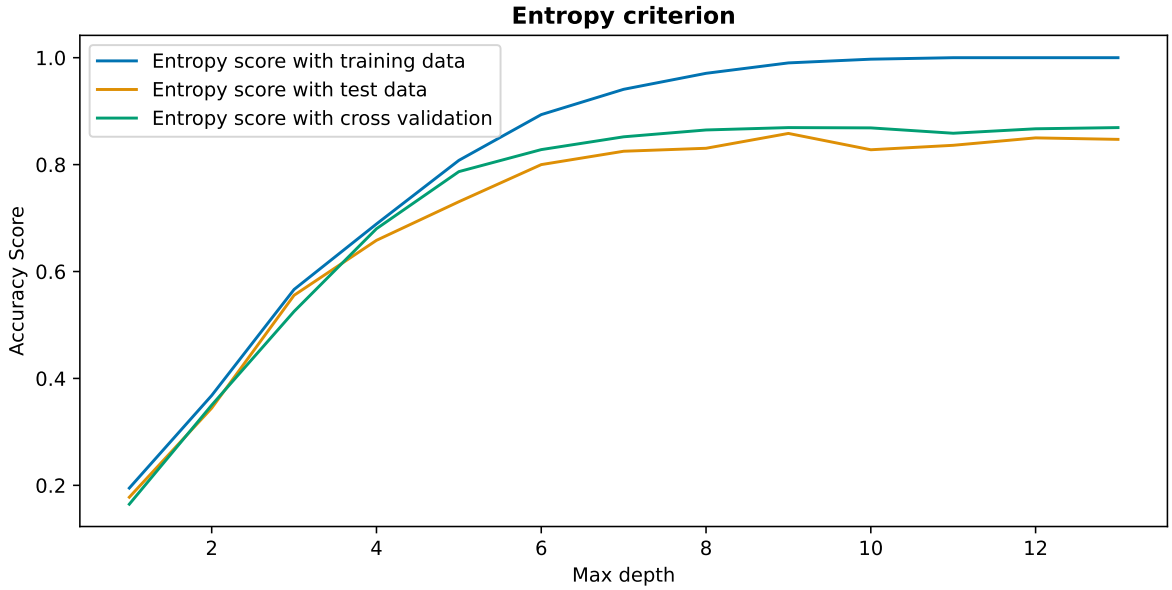
Q7)

On cherche maintenant à estimer le risque (ou de manière équivalente la précision) de manière plus précise en ayant recours à la cross-validation. Le principe consiste à répéter l'étape d'entraînement et d'estimation du risque sur plusieurs partitions en deux de notre jeu de données puis à prendre la moyenne des risques estimés. On va utiliser la fonction `skleanr.model_selection.cross_val_score` avec l'algorithme KFold.

```
classifiers = create_all_classifiers(criterions, depths)

scores_crossval = fmap_dict_list(
    lambda classifier: np.mean(model_selection.cross_val_score(
        classifier, X, y,
        cv = model_selection.KFold(5, shuffle=True, random_state=0)
    )),
    classifiers
)

plt.figure(figsize=(10, 10))
for i, criterion in enumerate(scores.keys()):
    plt.subplot(2, 1, i+1)
    plt.plot(
        depths, scores[criterion],
        label=f"{criterion.capitalize()} score with training data"
    )

    plt.plot(
        depths, scores_test[criterion],
        label=f"{criterion.capitalize()} score with test data"
    )

    plt.plot(
        depths, scores_crossval[criterion],
        label=f"{criterion.capitalize()} score with cross validation"
    )

    plt.title(f"{criterion.capitalize()} criterion", weight="bold")
    plt.xlabel("Max depth")
    plt.ylabel("Accuracy Score")
    plt.legend()
```

```
plt.draw()
```

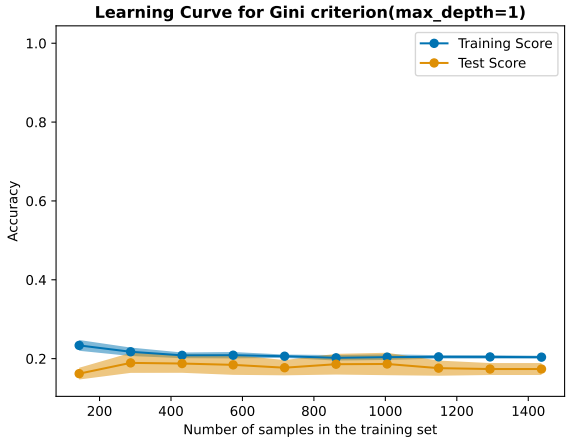


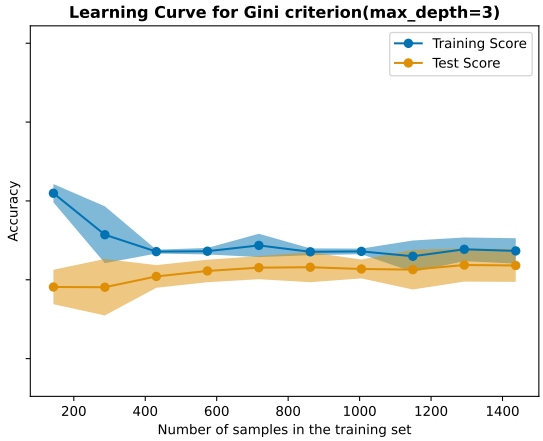


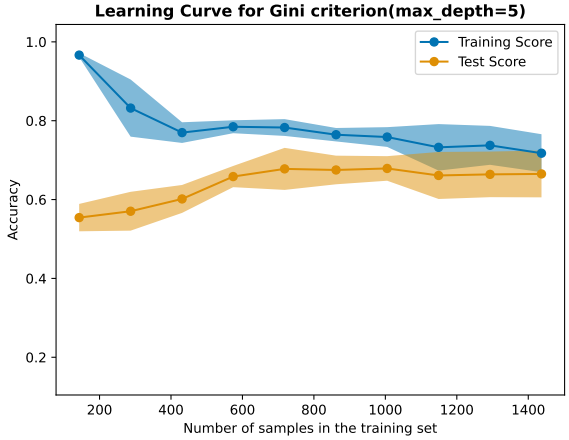
On remarque que l'estimation du risque via la cross-validation semble plus régulière que avec une seule estimation sur un jeu de test.

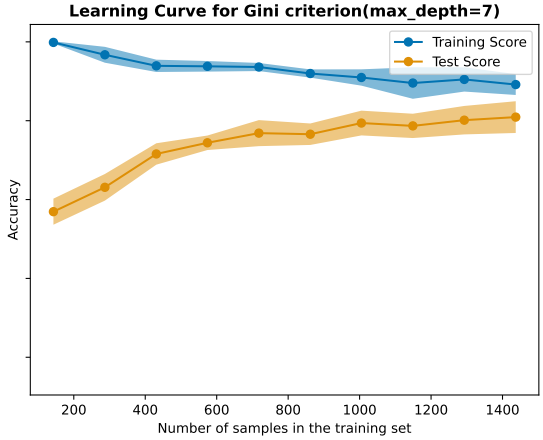
Q8)

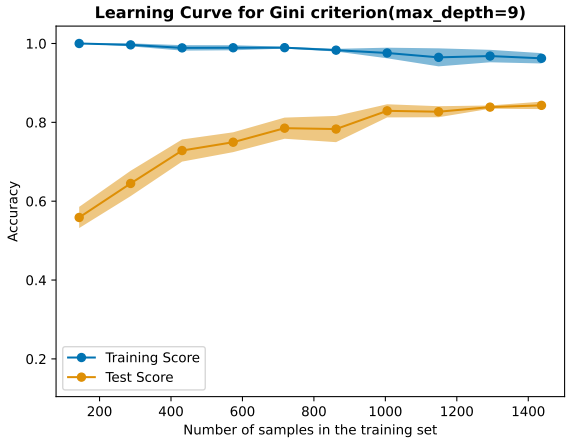
```
for criterion in classifiers.keys():
    fig, ax = plt.subplots(nrows=3, ncols=2, figsize=(15, 17), sharey=True)

    skip = 2
    for i, classifier in enumerate(classifiers[criterion][:12:skip]):
        sub_ax = ax[i//2, i%2]
        model_selection.LearningCurveDisplay.from_estimator(
            classifier,
            X = X, y = y,
            train_sizes = np.linspace(0.1, 1, 10),
            line_kw = {"marker": "o"},
            std_display_style = "fill_between",
            score_name = "Accuracy",
            ax = ax[i//2, i%2],
            cv = model_selection.KFold(5, shuffle=True, random_state=0)
        )

        sub_ax.set_title(
            (f"Learning Curve for {criterion.capitalize()} criterion"
            f"(max_depth={i * skip + 1})"),
            weight="bold"
        )

        sub_ax.legend(["Training Score", "Test Score"])
```

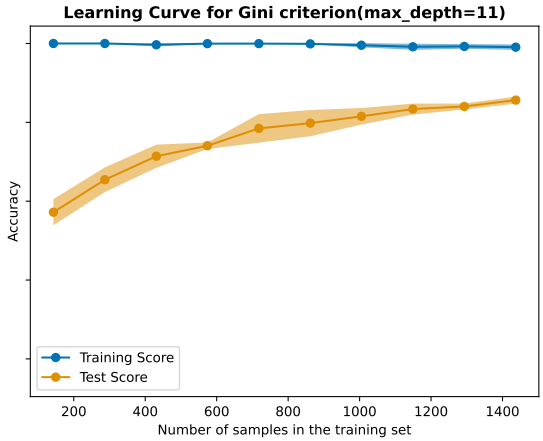

Learning Curve for Gini criterion(max_depth=1)
Training Score
Test Score
Accuracy
Number of samples in the training set

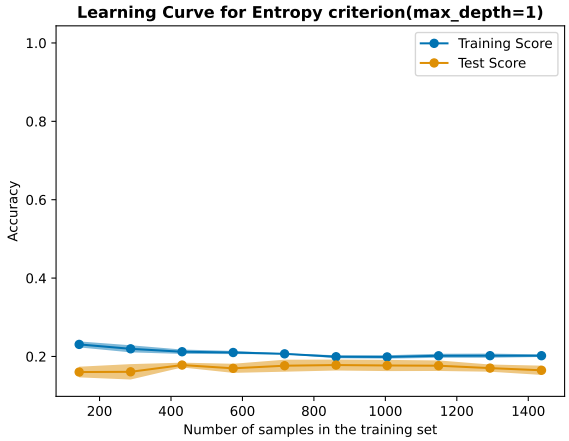

Learning Curve for Gini criterion(max_depth=3)
Training Score
Test Score
Accuracy
Number of samples in the training set

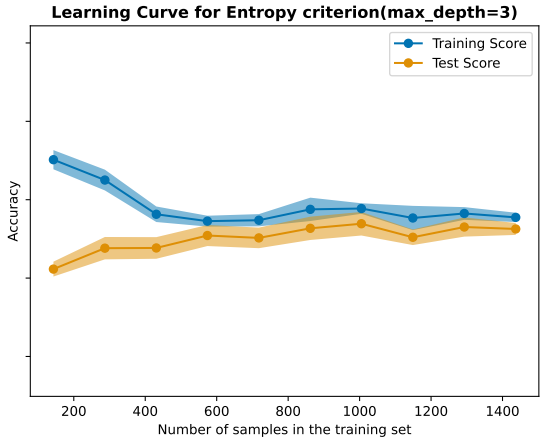

Learning Curve for Gini criterion(max_depth=5)
Training Score
Test Score
Accuracy
Number of samples in the training set

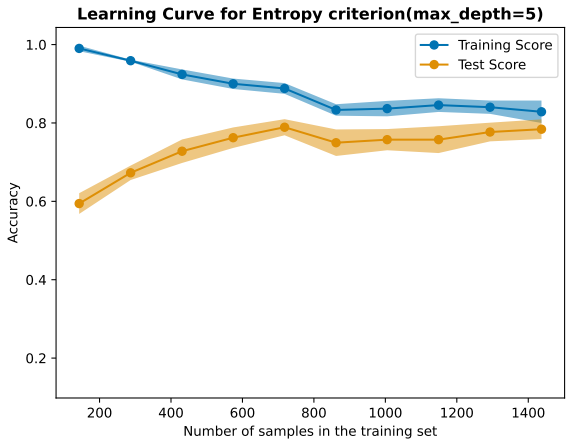

Learning Curve for Gini criterion(max_depth=7)
Training Score
Test Score
Accuracy
Number of samples in the training set

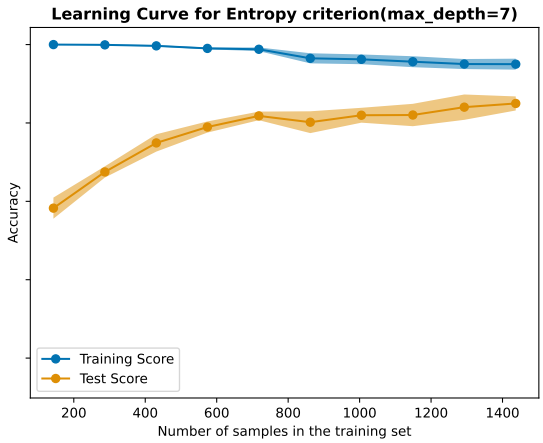

Learning Curve for Gini criterion(max_depth=9)
Training Score
Test Score
Accuracy
Number of samples in the training set

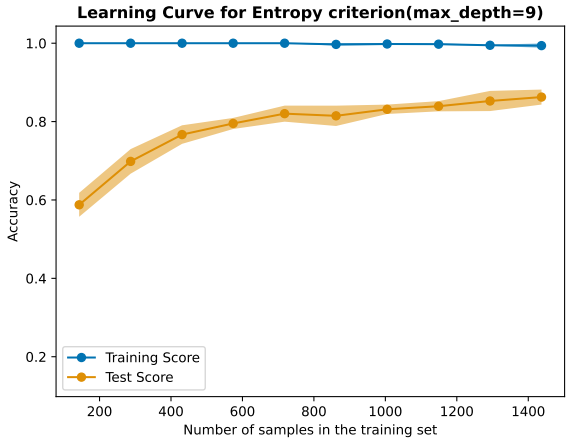

Learning Curve for Gini criterion(max_depth=11)
Training Score
Test Score
Accuracy
Number of samples in the training set

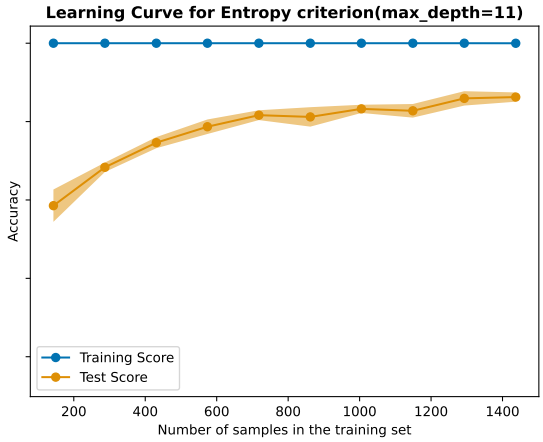

Learning Curve for Entropy criterion(max_depth=1)
Training Score
Test Score
Accuracy
Number of samples in the training set


Learning Curve for Entropy criterion(max_depth=3)
Training Score
Test Score
Accuracy
Number of samples in the training set


Learning Curve for Entropy criterion(max_depth=5)
Training Score
Test Score
Accuracy
Number of samples in the training set


Learning Curve for Entropy criterion(max_depth=7)
Training Score
Test Score
Accuracy
Number of samples in the training set


Learning Curve for Entropy criterion(max_depth=9)
Training Score
Test Score
Accuracy
Number of samples in the training set


Learning Curve for Entropy criterion(max_depth=11)
Training Score
Test Score
Accuracy
Number of samples in the training set