TP°2 : ARBRES

Lilou ZULEWSKI

2023-09-28

Table des Matières

L'objectif de ce TP est d'étudier les arbres de décision et la sélection de modèle, notamment dans le cardre de la classification.

1 Classification avec les arbres

1) **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 la dispersion des erreurs résiduelles entre les valeurs prédites par le modèle de régression et les valeurs réelles de la variable dépendante, à prédire. Sa valeur est obtenue en calculant la différence entre valeur réelle et 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`.

```
from sklearn import tree
```

2) **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.**

En veillant au bon équilibre des classes, on utilise la fonction `rand_checkers` pour effectuer une première simulation d'échantillons de taille $n = 456$.

Afin de créer des courbes donnant le pourcentage d'erreurs commises en fonction de la profondeur maximale de l'arbre, il faut d'abord construire nos arbres en utilisant soit le critère d'entropie soit celui de l'indice de Gini.

```
# construction des classifieurs

dt_entropy = tree.DecisionTreeClassifier(criterion="entropy")
dt_gini = tree.DecisionTreeClassifier(criterion="gini")

X = data[:,:2]
Y = np.array(data[:,2], dtype=int)
```

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

print("Critère d'Entropie :", dt_entropy.score(X,Y))
print("Paramètres du Classifieur Associé :", dt_entropy.get_params())
print("Critère de Gini :", dt_gini.score(X,Y))
print("Paramètres du Classifieur Associé :", dt_gini.get_params())
```

```
Critère d'Entropie : 1.0
Paramètres du Classifieur Associé : {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'en
Critère de Gini : 1.0
Paramètres du Classifieur Associé : {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gi
```

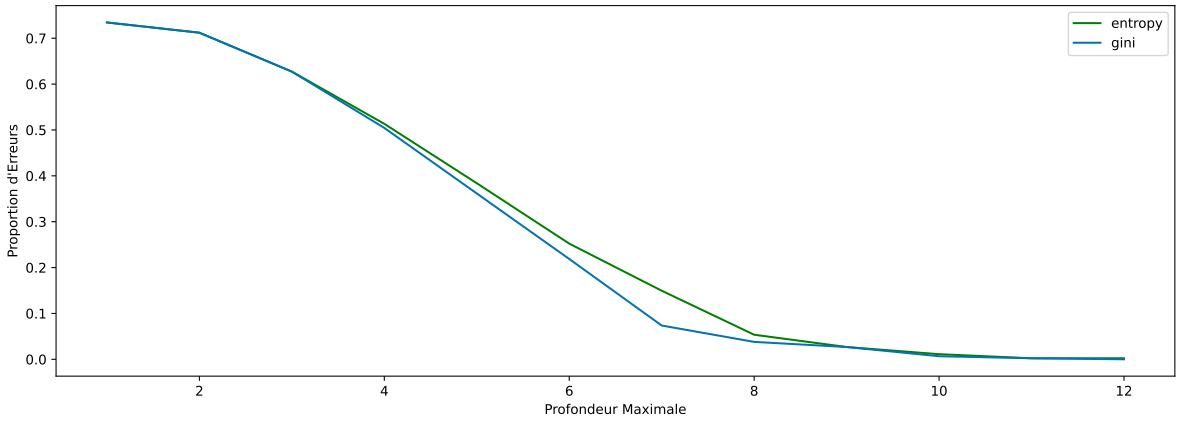
Dès lors, il est possible de construire le graphe d'erreurs commises en fonction de la profondeur maximale de l'arbre selon les deux critères.

```
# construction du graphe

d_max = 12
scores_entropy = np.zeros(d_max)
scores_gini = np.zeros(d_max)
for i in range(d_max):
    # critère d'entropie
    dt_entropy = tree.DecisionTreeClassifier(criterion='entropy', max_depth=i+1)
    dt_entropy.fit(X,Y)
    scores_entropy[i] = dt_entropy.score(X, Y)

    # critère de gini
    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(figsize=(15,5))
plt.plot(range(1, d_max + 1), 1-scores_entropy, label="entropy", color="green")
plt.plot(range(1, d_max + 1), 1-scores_gini, label="gini")
plt.legend()
plt.xlabel("Profondeur Maximale")
plt.ylabel("Proportion d'Erreurs")
plt.draw()
```



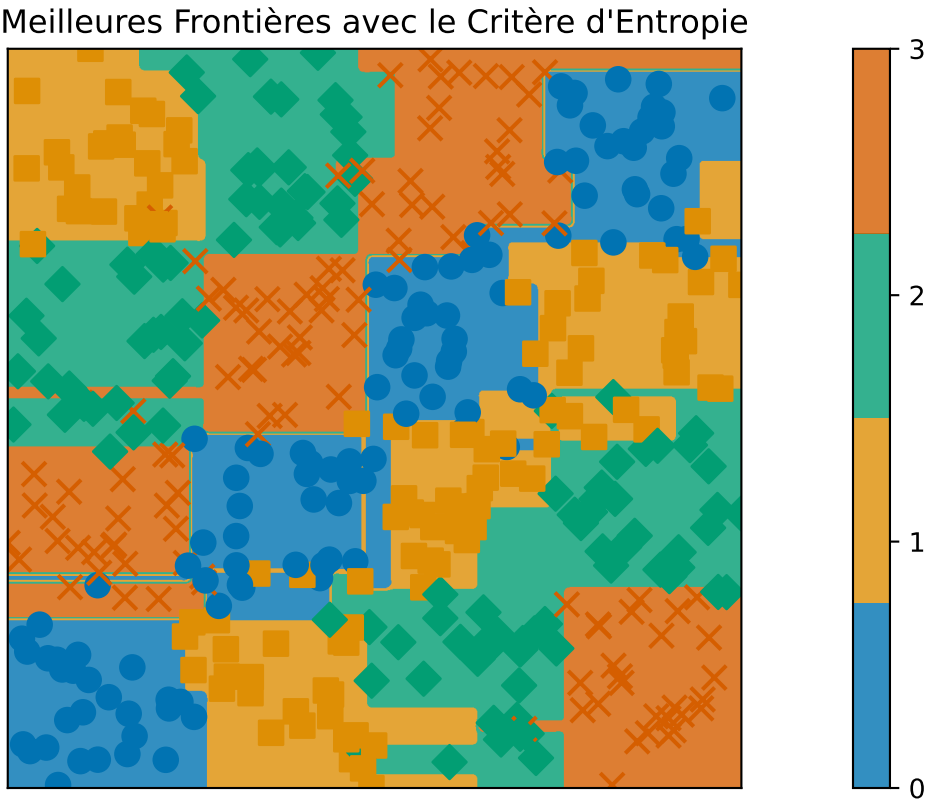
On remarque que les deux courbes se comportent de la même façon : le pourcentage d'erreur décroît pendant que la profondeur maximale de l'arbre croît. En particulier, il est proche de 0 lorsque la profondeur maximale de l'arbre dépasse 10. On peut en déduire que le modèle apprend et que ce classifieur est un bon classifieur.

3) **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).**

La classification obtenue en utilisant la profondeur qui minimise le pourcentage d'erreurs obtenues avec l'entropie peut être visualisée à l'aide de la fonction `frontiere` comme suit :

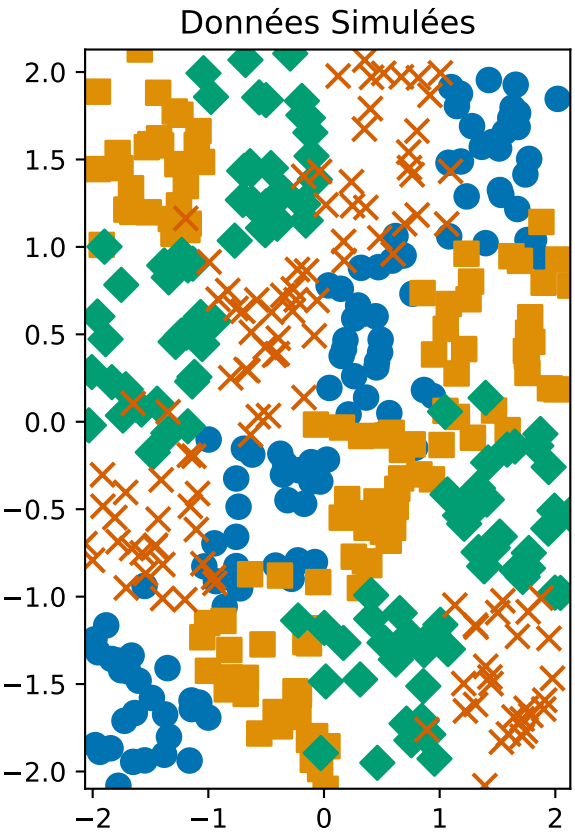
```
random.seed(2609)
# création de l'arbre avec la profondeur optimale pour l'entropie
dt_entropy.max_depth = np.argmin(1-scores_entropy)+1
# représentation graphique de la classification obtenue
plt.figure(figsize=(15,5))
frontiere(lambda x: dt_entropy.predict(x.reshape((1, -1))), X, Y, step=100)
plt.title("Meilleures Frontières avec le Critère d'Entropie")
plt.draw()
print("Meilleurs Scores avec le Critère d'Entropie :", dt_entropy.score(X, Y))
```

```
Meilleurs Scores avec le Critère d'Entropie : 0.9977678571428571
```



D'après le premier TP encadré sur les plus proches voisins, on sait que la fonction `rand_checkers` construit les données structurellement en quadrillage. La fonction `plot_2d` permet d'en avoir une visualisation :

```
# représentation graphique des données simulées
plt.ion()
plt.figure(figsize=(15, 5))
plt.subplot(141)
plt.title('Données Simulées')
plot_2d(data[:, :2], data[:, 2], w=None)
```



Il est intéressant de constater la répartition des données brutes comparée à la classification obtenue avec le critère d'entropie. En effet, les données simulées sont déjà plus ou moins partitionnées entre elles et semblent donc être idéales pour cette méthode de classification.

4) **Exporter un graphique de l'arbre obtenu à la question précédente en format pdf. On pourra par exemple utiliser la fonction `export_graphviz` du module `tree`.**

Afin de visualiser l'arbre de décisions obtenu à la question précédente, on utilise la fonction `export_graphviz` du module `tree` tel que :

```
import graphviz
tree.plot_tree(dt_entropy, filled=True)
data = tree.export_graphviz(dt_entropy, filled=True)
graph = graphviz.Source(data)
graph.render("./graphviz/dt_entropy", format='pdf')
```

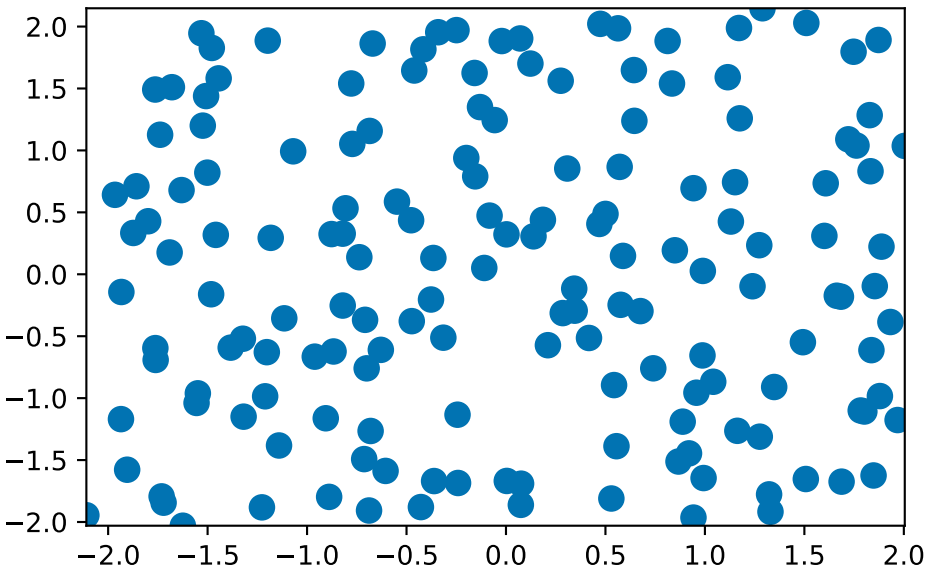
5) **Créez n = 160 = 40 + 40 + 40 + 40 nouvelles données avec `rand_checkers`. Pour les arbres de décision entraînés précédemment, calculer la proportion d'erreurs faites sur cet échantillon de test. Commenter.**

L'objectif est maintenant de créer un nouvel échantillon à l'aide de la fonction `rand_checkers` pour tester la validité de l'arbre obtenu précédemment en étudiant la proportion d'erreurs sur cet échantillon.

On commence par créer ce nouvel échantillon de test comme suit :

```
# génération de l'échantillon test
n = 160
n1 = n//4
n2 = n//4
n3 = n//4
n4 = n//4
sigma = 0.1
data_test = rand_checkers(n1, n2, n3, n4, sigma)
plot_2d(data_test)

X_test = data_test[:,:2]
Y_test = np.asarray(data_test[:,2], dtype=int)
```

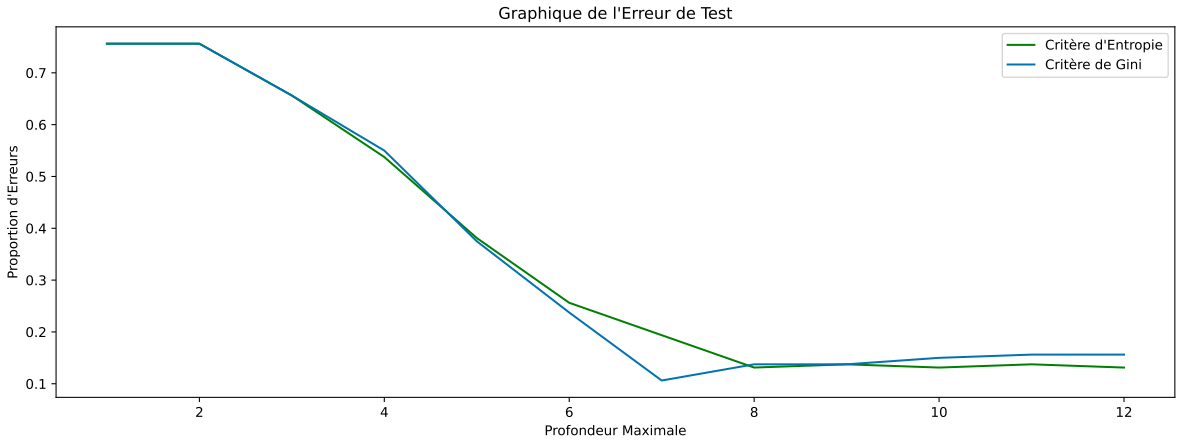


De la même façon que précédemment, on construit le graphe d'erreurs commises en fonction de la profondeur maximale de l'arbre selon les deux critères.

```
# construction du graphe
d_max = 12
scores_entropy = np.zeros(d_max)
scores_gini = np.zeros(d_max)
for i in range(d_max):
    # critère d'entropie
    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)
    # critère de gini
    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(figsize=(15,5))
plt.plot(range(1, d_max + 1), 1-scores_entropy, label="Critère d'Entropie", color="green")
plt.plot(range(1, d_max + 1), 1-scores_gini, label="Critère de Gini")
plt.legend()
plt.xlabel("Profondeur Maximale")
plt.ylabel("Proportion d'Erreurs")
plt.draw()
plt.title("Graphique de l'Erreur de Test")
```

```
Text(0.5, 1.0, "Graphique de l'Erreur de Test")
```



Il est alors possible de calculer la proportion moyenne d'erreurs faites sur cet échantillon de test.

```
# calcul de la proportion d'erreurs
error_rate_entropy = 1 - np.average(scores_entropy)
error_rate_gini = 1 - np.average(scores_gini)
print("Proportion Moyenne d'Erreurs avec le Critère d'Entropie :{:.2f}%".format(error_rate
print("Proportion Moyenne d'Erreurs avec le Critère de Gini :{:.2f}%".format(error_rate_gi
```

```
Proportion Moyenne d'Erreurs avec le Critère d'Entropie :35.05%
Proportion Moyenne d'Erreurs avec le Critère de Gini :34.79%
```

On constate que la proportion d'erreurs est légèrement plus élevée pour l'arbre de décisions basé sur le critère d'entropie que pour celui basé sur le critère de Gini. Ainsi, on peut supposer que la prédiction des échantillons est meilleure en terme de précision.

6) **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**

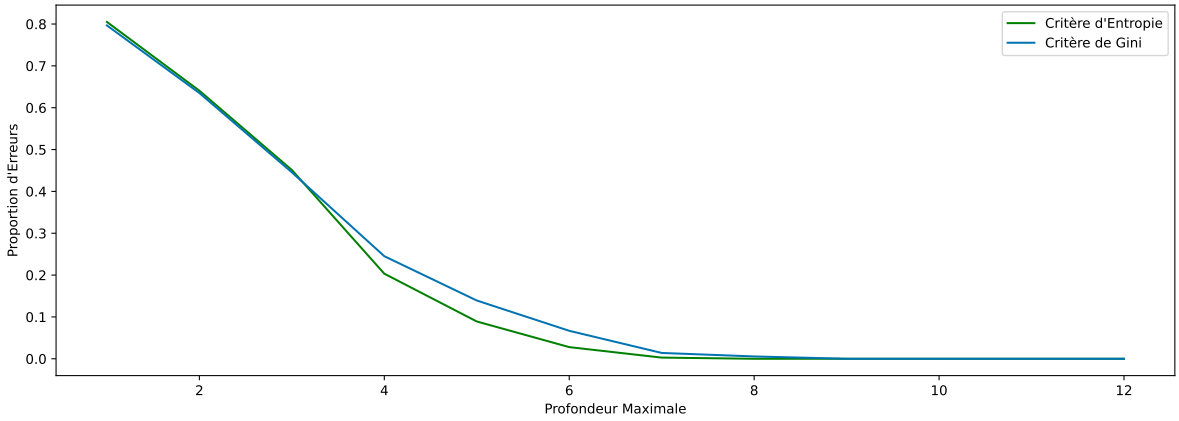
On va maintenant réitérer les questions précédentes pour le jeu de données `digits` présent dans le module `sklearn.datasets`. Ici, les données d'apprentissage et de test ne peuvent pas être simulées d'où le partitionnement du jeu de données suivant :

```
digits = datasets.load_digits()
n_samples = len(digits.data)
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(digits.data,
                                                                    digits.target,
                                                                    test_size=0.8,
                                                                    random_state=50)
```

On créée alors les deux courbes de pourcentage d'erreurs commises en fonction de la profondeur maximale de l'arbre pour les deux critères.

```
d_max = 12
scores_entropy = np.zeros(d_max)
scores_gini = np.zeros(d_max)
for i in range(d_max):
    # critère d'entropie
    dt_entropy = tree.DecisionTreeClassifier(criterion='entropy', max_depth=i+1)
    dt_entropy.fit(X_train,Y_train)
    scores_entropy[i] = dt_entropy.score(X_train, Y_train)
    # critère de gini
    dt_gini = tree.DecisionTreeClassifier(criterion='gini', max_depth=i+1)
    dt_gini.fit(X_train,Y_train)
```

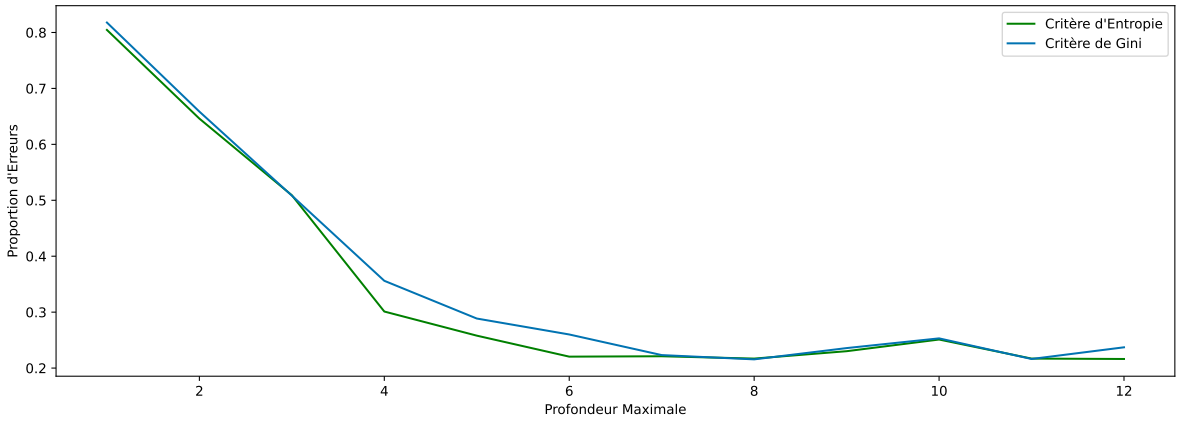
```
    scores_gini[i] = dt_gini.score(X_train,Y_train)
# affichage du graphique
plt.figure(figsize=(15,5))
plt.plot(range(1, d_max + 1), 1-scores_entropy, label="Critère d'Entropie", color="green")
plt.plot(range(1, d_max + 1), 1-scores_gini, label="Critère de Gini")
plt.legend()
plt.xlabel("Profondeur Maximale")
plt.ylabel("Proportion d'Erreurs")
plt.draw()
```



Or, l'objectif in fine est d'étudier les courbes d'erreurs sur l'échantillon test pour conclure sur la fiabilité de l'arbre de décisions.

```
d_max = 12
scores_entropy = np.zeros(d_max)
scores_gini = np.zeros(d_max)
for i in range(d_max):
    # critère d'entropie
    dt_entropy = tree.DecisionTreeClassifier(criterion='entropy', max_depth=i+1)
    dt_entropy.fit(X_train,Y_train)
    scores_entropy[i] = dt_entropy.score(X_test, Y_test)
    # critère de gini
    dt_gini = tree.DecisionTreeClassifier(criterion='gini', max_depth=i+1)
    dt_gini.fit(X_train,Y_train)
    scores_gini[i] = dt_gini.score(X_test,Y_test)
# affichage du graphique
plt.figure(figsize=(15,5))
plt.plot(range(1, d_max + 1), 1-scores_entropy, label="Critère d'Entropie", color="green")
```

```
plt.plot(range(1, d_max + 1), 1-scores_gini, label="Critère de Gini")
plt.legend()
plt.xlabel("Profondeur Maximale")
plt.ylabel("Proportion d'Erreurs")
plt.draw()
```



Pour le partitionnement choisi du jeu de données, les deux courbes ont la même allure générale. De plus, tout comme avec les données simulées, une décroissance remarquable de la proportion d'erreurs est présente pour les faibles valeurs de profondeur maximale ; cette proportion se stabilise ensuite pour des valeurs de profondeur maximale plus grandes.

2 Méthodes de choix de paramètres - Sélection de Modèle

7) **Utiliser la fonction `sklearn.model_selection.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.**

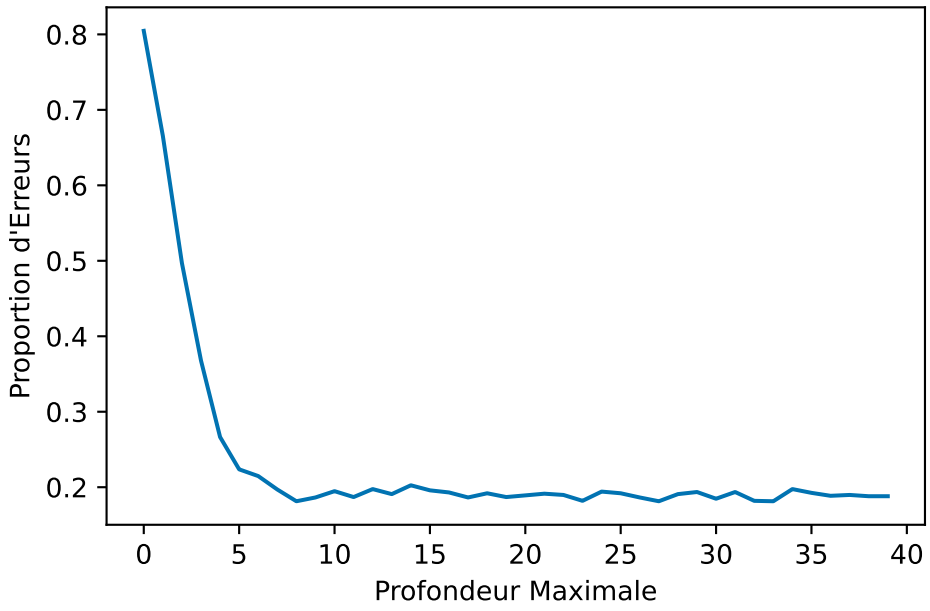
Cette seconde partie consiste à étudier la sélection de modèle par la validation croisée afin d'éviter d'avoir des résultats dépendant du partitionnement des données.

On va utiliser la fonction `sklearn.model_selection.cross_val_score`.

```
np.random.seed(2609)
# recherche de la profondeur optimale
d_max = 40
X = digits.data
Y = digits.target
```

```
error = np.zeros(d_max)
for i in range(d_max):
    dt_entropy = tree.DecisionTreeClassifier(criterion='entropy', max_depth=i+1)
    error[i] = np.mean(1 - cross_val_score(dt_entropy, X, Y, cv=5))
# affichage de la courbe d'erreurs
plt.plot(error)
plt.xlabel("Profondeur Maximale")
plt.ylabel("Proportion d'Erreurs")
# affichage de la profondeur optimale
d_optimal = 1 + np.argmin(error)
print("Profondeur Maximale Optimale: ", d_optimal)
```

```
Profondeur Maximale Optimale:  28
```



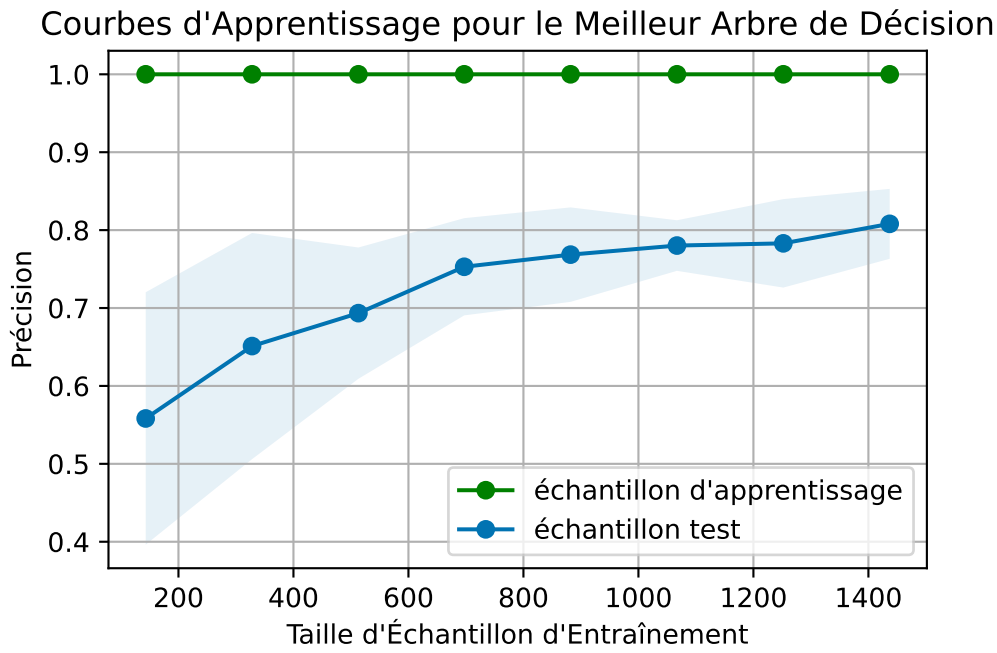
On remarque que la courbe d'erreur décroît jusqu'à se stabiliser aux alentours de 10. En effet, la proportion d'erreurs pour une profondeur de 30 est similaire à celle pour une profondeur de 15 par exemple. Selon l'algorithme précédent, la profondeur d'arbre minimisant la proportion d'erreurs moyenne est égale à 28.

8) **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.**

À partir de la ressource fournie, on trace la courbe d'apprentissage de l'arbre de décision obtenu à partir de la profondeur maximale optimale calculée précédemment.

```
# chargement de l'ensemble de données digits
digits = datasets.load_digits()
X = digits.data
Y = digits.target
# création d'un classificateur d'arbre de décision avec la profondeur maximale optimale
dt = tree.DecisionTreeClassifier(criterion='entropy', max_depth=28)
# calcul de la courbe d'apprentissage
n_samples, train_curve, test_curve = learning_curve(dt, X, Y,train_sizes=np.linspace(0.1,
# calculs des moyennes et écarts types des scores
train_scores_mean = np.mean(train_curve, axis=1)
test_scores_mean = np.mean(test_curve, axis=1)
train_scores_std = np.std(train_curve, axis=1)
test_scores_std = np.std(test_curve, axis=1)
# traçage de la courbe d'apprentissage
plt.figure()
plt.grid()
plt.fill_between(n_samples, train_scores_mean -1.96*train_scores_std,
                 train_scores_mean + 1.96*train_scores_std)
plt.fill_between(n_samples, test_scores_mean -1.96*test_scores_std,
                 test_scores_mean + 1.96*test_scores_std, alpha=0.1)
plt.plot(n_samples, train_scores_mean,"o-", label="échantillon d'apprentissage", color="gr
plt.plot(n_samples, test_scores_mean, "o-", label="échantillon test")
plt.legend(loc="lower right")
plt.xlabel("Taille d'Échantillon d'Entraînement")
plt.ylabel("Précision")
plt.title("Courbes d'Apprentissage pour le Meilleur Arbre de Décision")
```

```
Text(0.5, 1.0, "Courbes d'Apprentissage pour le Meilleur Arbre de Décision")
```



À partir de ce graphe de courbes d'apprentissage, on constate l'échantillon test possède une précision croissante avec la taille d'échantillon d'entraînement indiquant une bonne fiabilité du modèle.