Travaux pratiques de
Machine learning
Cycle pluridisciplinaire d'études supérieures
Université Paris sciences et lettres

Joon Kwon

vendredi 5 juin 2020

On considère à nouveau le jeu de données immobilières.

```
import numpy as np
from sklearn.datasets import load_boston

data = load_boston()
X = data.data
y = data.target

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
 ↪  train_test_split(X,y,random_state=17)
```

L'argument `random_state=17` permet de substituer la valeur 17 à chaque appel d'un générateur de nombre aléatoire lors de l'exécution de la fonction `train_test_split`. Cela permet de rendre la fonction déterministe et ainsi d'obtenir les mêmes échantillons d'apprentissage et de test à chaque exécution. Cela facilitera la comparaison des différents résultats, car les arbres de décision obtenus dans ce problème dépendent fortement de la composition de l'échantillon d'apprentissage.

On peut entraîner un arbre de décision pour ce problème de régression de la façon suivante.

```
from sklearn.tree import DecisionTreeRegressor
dt =
 ↪  DecisionTreeRegressor(random_state=0).fit(x_train,y_train)
```

Par défaut, le critère d'impureté employé en régression est l'erreur quadratique moyenne (MSE), et pour chaque nœud, la segmentation est choisie aléatoirement parmi celles minimisant le critère. Pour chaque arbre de décision entraîné dans ce TP, nous allons cependant imposer un comportement déterministe à l'aide de l'argument `random_state=0`. On peut accéder à la profondeur et au nombre de nœuds de l'arbre `dt` via `dt.tree_.max_depth` et `dt.tree_.node_count` respectivement.

QUESTION 1. — Définir une fonction `tree_summary` qui prend un argument un arbre de décision entraîné et qui affiche son score d'apprentissage, son score de test, sa profondeur et son nombre de nœuds. Appliquer cette fonction à l'arbre `dt` construit ci-dessus. Commenter les résultats.

On peut limiter la profondeur de l'arbre à l'aide de l'argument `max_depth`.

QUESTION 2. — Entraîner un arbre de décision, nommé `dt2`, dont la profondeur est limitée à 3. Observer ses scores et ses caractéristiques. Commenter.

On peut visualiser l'arbre de décision `dt2` de la façon suivante.

```
from sklearn.tree import plot_tree
plot_tree(dt2,fontsize=8, label=None, impurity=False)
```

QUESTION 3. — Quelles sont les variables explicatives qui interviennent dans l'arbre de décision `dt2` ?

QUESTION 4. — À l'aide d'une 10-validation croisée, choisir le meilleur paramètre pour la profondeur maximale de l'arbre de décision.

Les questions suivantes ont pour but la construction d'un prédicteur obtenu par *agrégation* de plusieurs arbres de décisions entraînés en faisant varier l'échantillon d'apprentissage. Plus précisément, en notant $S = (x_i, y_i)_{i\in[n]}$ l'échantillon d'apprentissage initial et m le nombre d'arbres de décision qu'on souhaite agréger, pour chaque $k \in [m]$, on tire un échantillon de taille n' : $S^{(k)} = (x_i^{(k)}, y_i^{(k)})_{i\in[n']}$ où chaque exemple $(x_i^{(k)}, y_i^{(k)})$ est tiré uniformément et indépendamment parmi

les exemples de S (il s'agit donc d'un tirage *avec remise*) ; on entraîne ensuite avec $S^{(k)}$ un arbre de décision $\hat{f}^{(k)}$; et le prédicteur final $\hat{f}$ est obtenu en moyennant :

$$\forall x \in \mathcal{X}, \quad \hat{f}(x) = \frac{1}{m} \sum_{k=1}^{m} \hat{f}^{(k)}(x).$$

Question 5. — Définir une fonction `r2_score` qui prend un argument deux arrays `y_true` et `y_predict` contenant respectivement, pour un échantillon donné, les vraies étiquettes et les étiquettes prédites par un prédicteur, et qui revoie le score R^2 correspondant.

```
from sklearn.utils import resample
```

Question 6. — À l'aide de la fonction `resample`, tirer (avec remise) à partir de l'échantillon d'apprentissage (`X_train`, `y_train`) un autre échantillon (`X_train_`, `y_train_`) de même taille (le tirage étant avec remise, certains exemples peuvent apparaître plusieurs fois dans le nouvel échantillon, d'autres vont en être absents). Entraîner à l'aide de ce nouvel échantillon un arbre de décision (avec `random_state=0` et les autres paramètres par défaut) et afficher son score R^2 en utilisant la fonction `r2_score` (on vérifiera qu'elle donne le même résultat que la fonction `.score` intégrée au prédicteur).

Question 7. — Construire 5 prédicteurs de la même façon qu'à la question précédente en ré-échantillonnant (`X_train_`, `y_train_`) à chaque fois. Calculer les prédictions des prédicteurs obtenus sur l'échantillon de test.

Question 8. — On considère le prédicteur défini comme la moyenne des 5 prédicteurs construits à la question précédente. Calculer ses prédictions sur l'échantillon de test et calculer le score R^2 correspondant.

Question 9. — Définir une fonction `tree_aggregation` qui prend en argument un entier `n_trees`, qui entraîne `n_trees` arbres de décision comme précédemment, et qui affiche le score R^2 du prédicteur agrégé sur l'échantillon de test. Exécuter cette fonction avec différentes valeurs pour `n_trees` et tenter d'obtenir un meilleur score.

La procédure qui consiste à construire un nouvel échantillon d'apprentissage à l'aide de tirages uniformes avec remise s'appelle le *bootstrap*. Celle qui consiste

à agréger plusieurs prédicteurs obtenus par bootstrap s'appelle le *bagging* (pour *bootstrap aggregating*). Enfin, dans le cas particulier des arbres de décision, le *bagging* est également appelé *forêt aléatoire*. `Scikit-learn` propose un algorithme de forêt aléatoire prêt-à-l'emploi qu'on peut utiliser comme suit.

```
from sklearn.ensemble import RandomForestRegressor
rf =
 ↪  RandomForestRegressor(n_estimators=5).fit(x_train,y_train)
```

Question 10. — Comparer les résultats obtenus à l'aide de la fonction `tree_aggregation` et ceux de l'algorithme fourni par `scikit-learn`.