TP

Apprentissage automatique

PSI : Lycée Rabelais

1 Présentation du problème

On s'intéresse à la prédiction du prix d'un bien immobilier en connaissant un certain nombre de critères. La base de données disponible a été établie dans les années 1990 dans le cadre de travaux de recherche. Elle s'intéresse aux biens immobiliers en Californie. Cette base est composée des caractéristiques suivantes en entrée :

- `MedInc` : le revenu médian des habitants du quartier ;
- `HouseAge` : l'âge médian des biens immobiliers du quartier ;
- `AveRooms` : le nombre de pièces dans le bien ;
- `AveBedrms` : le nombre de chambres à coucher ;
- `Population` : la population du quartier ;
- `AveOccup` : le nombre de personnes habitant le foyer ;
- `Latitude` : la latitude du quartier ;
- `Longitude` : la longitude du quartier.

La sortie est le prix médian du bien immobilier exprimé en centaine de milliers de dollars.

L'objectif est de définir un modèle permettant de prédire le prix du bien en fonction des caractéristiques préalablement mentionnées.

La trame du code est disponible sur votre *cahier-de-prépa* sous le nom `prix_immobilier___eleve.py`.

Question 1. Ouvrir, tester le code et commenter (ci-dessous) les principales instructions.

```
import numpy as np

from sklearn.datasets import fetch_california_housing
california_housing = fetch_california_housing()

from sklearn import datasets
prix = datasets.fetch_california_housing() ## importation de la bibliothèque

y = prix.target ## y contient le prix des biens immobiliers
x = prix.data ## X contient le tableau des données d'entrée

import seaborn as sns
import matplotlib.pyplot as plt

def trace_loc(x,y,T): ## cette fonction permet de tracer
                      ## le prix NORMALISé en fonction de la localité
                      ## NOTA : T permet d'afficher un commentaire
    plt.figure()
    ymap = (y - y.min())/ (y.max() - y.min())
    sns.scatterplot(x[:,-2],x[:,-1],ymap,cmap="viridis")
    plt.xlabel('Longitude (en degrés)')
    plt.ylabel('Latitude (en degrés)')
    plt.title("Prix normalisé de l'immobilier (1 = prix max ; 0 = prix min)"+'\n'+T)
    plt.grid(True)

```

```
def trace_simple(x,i,y,T): ## cette fonction permet de tracer
                           ## le prix en fonction de la i-ème caractéristique
                           ## NOTA : T permet d'afficher un commentaire
    plt.figure()
    plt.plot(x[:,i],y,'.')
    features = ["du revenu médian des habitants du quartier","de l'age médian des biens
    immobiliers du quartier","du nombre de pièces dans le bien","du nombre de chambres à
    coucher","de la population du quartier","du nombre de personnes habitant le foyer","de la
    latitude du quartier","de la longitude du quartier"]
    plt.xlabel('En fonction '+features[i])
    plt.ylabel('Prix en centaine de milliers de dollars')
    plt.title("Prix de l'immobilier en fonction "+features[i]+'\n'+T)
    plt.grid(True)


##################################
############ création du modèle de régression et exportation des résultats
##################################
def apprentissage():
    ## pour importer MLPRegressor
    from sklearn.neural_network import MLPRegressor
    mlp = MLPRegressor(activation='identity',hidden_layer_sizes=(4,2)) ## Création du modèle

    from sklearn.model_selection import train_test_split
    ## Préparation des données
    x_train, x_test, y_train, y_test = train_test_split(x, y,shuffle=True, test_size=0.5)
    ## nota : test_size=0.5 permet d'avoir autant de données de test que de données d'
    apprentissage

    mlp.fit(x_train,y_train)
    y_pred = mlp.predict(x_test)

    return(x_train, x_test, y_train, y_test,y_pred,mlp) ## Score obtenu


##################################
############ récupération des résultats
##################################
x_train, x_test, y_train, y_test,y_pred,mlp = apprentissage()
score = mlp.score(x_test, y_test)
trace_loc(x_train,y_train,"!!!! Tracé des données d'apprentissage !!!!")
trace_loc(x_test,y_pred,"!!!! Tracé des données prédites !!!!")
trace_simple(x_train,0,y_train,"!!!! Tracé des données d'apprentissage !!!!")
print("prix minimal prédit",min(y_pred))
print("prix maximal prédit",max(y_pred))
print("prix poyen prédit",np.mean(y_pred))
```

Question 2. Combien de paramètres (poids et biais) sont présents dans le modèle ? Le nombre de données est-il pertinent ?

Question 3. Utiliser la fonction `trace_simple` (sans relancer entièrement la phase d'apprentissage) pour visualiser l'influence de chacune des caractéristiques d'entrée sur le prix de l'immobilier.

Question 4. Donner une unité à chacune des caractéristiques d'entrée.

Un problème récurrent concerne la gestion des entrées lorsque celles-ci ne représentent pas la même grandeur.

On peut constater qu'une entrée joue un rôle prépondérant, si celle-ci est, en valeur numérique, plus grande que les

autres. Pour éviter ce problème, une étape fréquemment utilisée est donc la normalisation des données. Il s'agit, par exemple, de calculer pour une entrée x :

$$x_{\text{normalisée}} = \frac{x}{x_{\max}}$$

Ce problème étant fréquent, des modules adaptés permettent la normalisation de tout le tableau des entrées.

Question 5. Rajouter les lignes suivantes, permettant la normalisation, avant la phase d'apprentissage et observer l'effet sur le score de l'algorithme.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)
```

Question 6. Commenter les nuages de points obtenus et notamment :

- d'un point de vue de la géographie (on pourra chercher une carte de la Californie sur internet si besoin).
- en comparant les données d'apprentissage et celles prédites.

Question 7. Écrire une fonction permettant de calculer le score moyen sur n_{essai} essais.

Question 8. Commenter les résultats obtenues d'un point de vue du score et en visualisant les résultats. Les prédictions semblent-elles pertinentes ?

La documentation indique :

> `score` : Return the coefficient of determination R^2 of the prediction.
>
> The coefficient of determination R^2 is defined as $1 - \frac{u}{v}$, where u is the residual sum of squares `((y_true - y_pred)**2).sum()` and v is the total sum of squares `((y_true - y_true.mean()) ** 2).sum()`. The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of `y`, disregarding the input features, would get a score of 0.0.

On peut également évaluer l'intérêt du réglage du taux d'apprentissage (*learning rate*) en rajoutant dans le modèle :

```
mlp = MLPRegressor(activation='identity',hidden_layer_sizes=(10,10),learning_rate_init=0.001)
```

(ici on impose un taux d'apprentissage de 0.001)

On peut également afficher le temps d'apprentissage des paramètres en modifiant dans le code précédent :

```
from time import time
start = time()
apprentissage()
t_app = time()- start
```

Question 9. Faire quelques simulations (environ 5 pour chaque taux d'apprentissage) et remplir le tableau ci-dessous puis conclure.

Taux d'apprentissage	Temps d'apprentissage moyen (en s)	Score moyen
0.001		
0.01		
0.1		
0.5		

Question 10. Le score est-il toujours un indicateur objectif ?

Question 11. Modifier le code pour que celui-ci ne prenne en compte que la latitude et la longitude du bien immobilier (on rappelle que, pour un tableau `numpy` de dimension $n \times m$ noté `TAB`, `TAB[:,i:j]` renvoie toutes les lignes et les colonnes allant de la i-ième (incluse) à la j-ième exclue).

Question 12. Observer le résultat puis commenter les lignes rajoutées pour ne plus les prendre en compte dans la suite.

2 Amélioration du modèle

Dans le modèle préalablement défini, on a choisi une fonction identité pour la fonction d'activation.

Question 13. Modifier cette fonction d'activation en imposant une fonction sigmoïde (`logistic`). Observer le score et conclure.

Question 14. Modifier la structure du réseau en imposant le nombre de neurones sur chaque couche cachée en écrivant, par exemple, `hidden_layer_sizes=(40,20,10)` puis ensuite `hidden_layer_sizes=(2)`. Cela signifierait, dans le premier exemple, qu'il y a trois couches cachées de 40 puis 20 puis 10 neurones. Déterminer le nombre de paramètres associés à chaque configuration. Observer l'effet sur le score obtenu.

Question 15. Quelle est la condition principale pour avoir un modèle très performant ?

Question 16. Définir le réseau le plus performant. **Attention, votre réseau ne doit pas surapprendre !**

3 Utilisation d'un algorithme des k-plus proches voisins

Question 17. Rappeler le fonctionnement d'un tel algorithme.

Question 18. Modifier le modèle en écrivant :

```
from sklearn.neighbors import KNeighborsRegressor
mlp = KNeighborsRegressor(n_neighbors=3) ## Création du modèle
```

Question 19. Observer le score. Choisir le nombre de voisins pour que votre modèle soit le plus performant possible.