Intelligence artificielle et Apprentissage
TP : régression non linéaire (et non paramétrique)
Version Python

F. Lauer

1 Régression non linéaire

Nous allons traiter un problème de régression non linéaire dans lequel la fonction recherchée, c'est-à-dire le modèle optimal (aussi appelé fonction de régression), est le sinus cardinal :

$$\forall x \in \mathbb{R}, \quad f_{reg}(x) = \text{sinc}(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x}, & \text{si } x \neq 0 \\ 1, & \text{sinon} \end{cases}$$

Les points suivants sont à réaliser dans `tpregression.py` à récupérer sur ARCHE.

1. Générez un jeu de 1000 données pour X distribué uniformément dans $[-3, 3]$ et $Y = \text{sinc}(X) + V$, où V est un bruit gaussien centré de variance 0.2^2.
2. Découper ce jeu de données en une base d'apprentissage de 30 exemples et une base de test contenant les autres exemples. Affichez tous ces points sur un même graphique avec des étoiles pour la base d'apprentissage et des points pour la base de test. Tracez également la fonction de régression (par exemple en l'évaluant sur la base de test).

2 Méthodes non paramétriques pour la régression non linéaire

Les méthodes non paramétriques permettent d'apprendre des fonctions non linéaires assez complexes sans connaissance a priori sur la forme du modèle recherché. En effet, celles-ci considèrent que non seulement les paramétres du modèle sont à apprendre à partir des données mais aussi sa structure.

1. Complétez la fonction `krrapp(Xapp, Yapp, Lambda, sigma)` pour implémenter l'algorithme d'apprentissage de la *Kernel ridge regression* avec un noyau gaussien en utilisant la fonction

   ```
   K = kernel(Xapp, Xapp, sigma)
   ```

   pour calculer la matrice de noyau

   $$\texttt{K} = \boldsymbol{K} = \begin{bmatrix} K(x_1, x_1) & \dots & K(x_1, x_m) \\ \vdots & & \\ K(x_m, x_1) & \dots & K(x_m, x_m) \end{bmatrix}$$

   Implémentez aussi la fonction `krrpred` pour calculer les prédictions avec un tel modèle. Pour cela, vous pourrez utiliser

   ```
   Ktest = kernel(Xtest, Xapp, sigma)
   ```

pour calculer la matrice

$$\boldsymbol{K}_{test} = \begin{bmatrix} K(x_1^{test}, x_1) & \dots & K(x_1^{test}, x_m) \\ \vdots & & \\ K(x_n^{test}, x_1) & \dots & K(x_n^{test}, x_m) \end{bmatrix}$$

permettant de calculer les prédictions sur des points de test avec

$$\begin{bmatrix} f(x_1^{test}) \\ \vdots \\ f(x_n^{test}) \end{bmatrix} = \boldsymbol{K}_{test}\boldsymbol{\beta}$$

2. Testez la méthode KRR sur les données du sinus cardinal pour différentes valeurs de σ et λ et étudiez l'influence de ces hyperparamètres sur la forme du modèle obtenu et sur les erreurs d'apprentissage et de test (qu'il faudra donc calculer).

3. Pour vérifier, comparez les résultats à ceux obtenus en utilisant la bibliothèque scikit-learn qui permet de créer un modèle KRR avec

```
from sklearn.kernel_ridge import KernelRidge

model = KernelRidge(alpha = Lambda, kernel='rbf', gamma = 1/(2*sigma*sigma))
model.fit(Xapp, Yapp)
ypred = model.predict(Xtest)
```

Le paramètre `alpha` correspond à λ, le noyau `'rbf'` est le noyau gaussien $K(x, x') = \exp(-\gamma||x - x'||^2)$ correspondant à notre définition pour $\gamma = 1/2\sigma^2$.

Si les x_i sont en dimension 1, les matrices `Xapp` et `Xtest` sont en fait des vecteurs (tableaux à 1 dimension). Dans ce cas, il faut utiliser la fonction `Xapp.reshape(-1,1)` pour convertir le vecteur `Xapp` en matrice à une seule colonne, c'est-à-dire un tableau à deux dimensions attendu par les fonctions `.fit` et `.predict` habituées à travailler en plus grande dimension que 1.

2.1 K-plus proches voisins pour la régression

Essayez de voir comment adapter à la régression l'algorithme des K-plus proches voisins déjà utilisé pour la classification.

1. Complétez la fonction `kppvreg(Xtest,Xapp,Yapp, K)` pour implémenter l'algorithme des K-plus proches voisins pour la régression.

2. Testez cet algorithme sur les données du sinus cardinal pour différentes valeurs de K et étudiez l'influence de cet hyperparamètre sur la forme du modèle obtenu.

3. Testez aussi avec la fonction fournie par scikit-learn :

```
from sklearn import neighbors

kppv = neighbors.KNeighborsRegressor( K )
kppv.fit(Xapp.reshape(-1,1), Yapp)
ypred = kppv.predict(Xtest.reshape(-1,1))
```