Intelligence Artificielle & Machine Learning
TP régression et surapprentissage

Pré-requis

Pour réaliser des calculs scientifiques (par exemples sur des matrices) en Python, nous utilisons la bibliothèque `numpy`, ainsi qu'une autre qui permet de tracer des figures dans un environnement similaire à Matlab :

```
import numpy as np
import matplotlib.pyplot as plt
```

Pour installer ces bibliothèques sous Ubuntu :

```
sudo apt install python3-numpy python3-scipy python3-matplotlib
```

1 Régression linéaire

Nous allons implémenter en Python la méthode des moindres carrés pour la régression linéaire. Celle-ci correspond à l'application directe du principe ERM pour une classe de fonctions linéaires

$$\mathcal{F} = \left\{ f \mid f(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{w} \rangle = \boldsymbol{x}^T \boldsymbol{w} = \sum_{j=1}^{d} w_j x_j,\ \boldsymbol{w} \in \mathbb{R}^d,\ b \in \mathbb{R} \right\}$$

1. Générer un jeu de $m = 1000$ données $\boldsymbol{x}_i$ en dimension $d = 5$ dans la matrice

   $$\mathtt{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_m^T \end{bmatrix} \in \mathbb{R}^{m \times d}$$

   comme des réalisations indépendantes de la variable aléatoire X uniformément distribuée sur $[-5, 5]^d$.

2. Générez le vecteur des étiquettes correspondantes

   $$\mathtt{Y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \in \mathbb{R}^{m \times 1}$$

   sachant que la loi jointe du couple (X, Y) est telle que

   $$Y = \langle \boldsymbol{w}^*, X \rangle + V,$$

   avec

   $$\boldsymbol{w}^* = \begin{bmatrix} 1,2 \\ -1,3 \\ 0.5 \\ 0.8 \\ -2.3 \end{bmatrix}$$

et V un bruit gaussien de moyenne nulle et de variance 1.
Autrement dit, Y est une fonction linéaire de X de paramètre $\boldsymbol{w}^*$ perturbée par un bruit aléatoire V.
Commandes utiles : `np.random.rand` *et* `np.random.randn`.
Attention à ne pas confondre les variables aléatoires X et Y avec les matrices/vecteurs `X` *et* `Y`

3. En régression linéaire, la capacité du modèle à prédire correctement dépend de la proximité du vecteur de paramètres $\boldsymbol{w}$ au vecteur optimal $\boldsymbol{w}^*$.
Nous allons ici estimer $\boldsymbol{w}^*$ par la méthode des moindres carrés, c'est-à-dire en minimisant l'erreur quadratique moyenne sur l'ensemble des données :

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}\in\mathbb{R}^d} \sum_{i=1}^{m} (y_i - \boldsymbol{x}_i^T \boldsymbol{w})^2$$

Programmez la solution analytique de ce problème en fonction de `X` et `Y` (avec la fonction `np.linalg.inv` pour l'inversion matricielle). Vérifiez que cela donne une bonne estimation des paramètres du modèle linéaire et calculez l'erreur d'apprentissage pour le modèle $f(\boldsymbol{x}) = \boldsymbol{x}^T \hat{\boldsymbol{w}}$.

4. La solution des moindres carrés peut aussi s'obtenir simplement et de manière plus robuste avec la fonction dédiée :

```
res = np.linalg.lstsq(X,Y)
what = res[0]
```

Comparez les résultats de cette méthode avec la précédente.

5. Testez avec différentes valeurs de m pour observer l'amélioration de l'estimation mesurée par $\|\boldsymbol{w}^* - \hat{\boldsymbol{w}}\|$ avec l'augmentation du nombre d'exemples.

6. Modifiez la matrice `X` pour que la dernière colonne ne soit plus aléatoire mais égale à 2 fois la 4ème et relancez l'expérience. Que donnent les différentes approches dans ce cas ? Pourquoi ?

2 Régression polynomiale

Le problème d'apprentissage précédent était assez simple : le modèle optimal (aussi appelé fonction de régression) $f_{reg}(\boldsymbol{x}) = \langle \boldsymbol{w}^*, \boldsymbol{x} \rangle$ faisait partie de $\mathcal{F}$ et le nombre de paramètres à estimer était très faible devant le nombre de données.

Nous allons maintenant traiter un problème de régression non linéaire dans lequel la fonction recherchée est le sinus cardinal :

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

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

1. Générez un jeu de 1000 données pour X distribuée uniformément dans $[-3, 3]$ et $Y = \mathrm{sinc}(X) + V$, 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).

3. Nous supposons la forme de la fonction recherchée comme inconnue et choisissons d'apprendre un modèle polynomial de degré D de la forme

$$f(x) = \sum_{k=0}^{D} w_k x^k$$

Proposez une procédure pour appliquer la méthode des moindres carrés ici et implémentez-la dans la fonction `polyreg.m`. Implémentez également la fonction `polypred.m` pour calculer $f(x)$ quelque soit x.

4. Appliquez ces fonctions dans le programme principal pour apprendre f, tracer la courbe de $f(x)$ et évaluer les erreurs d'apprentissage et de test pour $D = 5$.

5. Le degré du polynôme est un hyperparamètre de la méthode (paramétrant la classe de fonctions $\mathcal{F}$). Réalisez les apprentissages, tracés et calculs d'erreurs pour tous les degrés D entre 1 et 15 et déterminez le degré optimal.

6. Quel phénomène est mis en avant ici pour les grandes valeurs de D ?

7. Le choix du meilleur D précédent s'appuie sur l'erreur de test. En quoi cela n'est-il pas convenable ? Proposez une procédure correcte pour la sélection de modèle (choix de D) et l'estimation du risque du modèle sélectionné.

8. Comment faire la sélection de modèle sans utiliser d'autres données que les 30 points de la base d'apprentissage ?