TP1 - ML
Régression linéaire

6 février 2025

Le but de cette séance consiste à se familiariser avec `Python` et `Pytorch` en se basant sur un poblème relativement simple. On considère un jeu de données $X_i \in \mathbb{R}^n$ ($i \in \{1, \cdots, N\}$) dont dépend une variable $y_i \in \mathbb{R}$. L'objectif consiste à déterminer la dépendance de la valeur y_i en fonction des données X_i. Afin de simplifier l'analyse, nous allons construire arbitrairement un jeu de données, puis dans un second temps utiliser différents algorithmes afin de déterminer la dépendance en question.

Exercise 1 Cas scalaire

On considère le cas où les données sont scalaires, c'est à dire $n = 1$ et $X_i = x_i \in \mathbb{R}$. On suppose également une dépendance affine des variables y_i en fonction des données, c'est à dire qu'il existe une poids w et un biais α tels que pour tout $i \in \{1, \cdots, N\}$,

$$y_i = wx_i + \alpha.$$

On souhaite déterminer simplement à partir des données observées x_i et y_i les poids et biais α et w.

1. Montrer que'il suffit de deux observations distinctes (i.e. $N = 2$) pour déterminer α et w. Donner une formule explicite qui permet de calculer w et α en fonction de x_1, x_2, y_1 et y_2.

2. Dans la pratique, les données sont bruitées, c'est à dire que

$$y_i = wx_i + \alpha + b_i, \tag{1}$$

où b_i est un bruit aléatoire (i.e. une variable aléatoire). Dans ce cas, les points (x_i, y_i) ne sont pas alignés parfaitement le long d'une droite, mais s'en écarte légèrement. Définir une fonction `noise` qui prend en argument σ, une variance, et simule le bruit gaussien centré correpsondant.

Pour rappel, si U_1 et U_2 sont deux variables aléatoires indépendantes uniformément distribuées sur $(0, 1)$, la variable aléatoire

$$b = \sigma\sqrt{-2\ln(U_1)}\cos(2\pi U_2)$$

défini un bruit gaussien centré de variance σ comme demandé.

3. À partir des données, on souhaite retrouver le plus précisemment possible les poids w et α. À cet effet, on introduit la fonction coût

$$F(w, \alpha) = \sum_{i=1}^{N} |y_i - wx_i - \alpha|^2.$$

Définir la fonction `F` qui prend en argument les vecteurs x et y, ainsi que les paramètres w et α et renvoie $F(w, \alpha)$.

4. Définir une fonction `generationWalpha` qui renvoie un poids et un biais w et α tirés aléatoirement, uniformément et indépendamment dans $[-1, 1]$.

5. Définir une `generationx` qui prend en argument un entier n et génére un vecteur de $\mathbb{R}^n$ dont chaque coordonnée est tirées aléatoirement, uniformément et indépendamment dans $[-1, 1]$.

6. Définir une fonction `generationy` qui prend en argument un vecteur $x \in \mathbb{R}^n$, une variance σ, un poids w et un biais α et renvoie un vecteur y comme défini par (1).

7. Montrer que (w, α) est solution du problème de minimisation

$$\min_{w,\alpha} F(w, \alpha)$$

si et seulement si

$$\left\{ \begin{array}{l} \sum_{i=1}^{N}(y_i - wx_i - \alpha)x_i = 0 \\ \sum_{i=1}^{N}(y_i - wx_i - \alpha) = 0. \end{array} \right. \tag{2}$$

8. Montrer que le système (2) est équivalent à

$$w \sum_{i=1}^{N} (x_i - \overline{x})^2 = \sum_{i=1}^{N} (x_i - \overline{x}) y_i$$

et

$$\alpha = \frac{1}{N} \sum_{i=1}^{N} (y_i - w x_i).$$

En déduire que le système (2) admet au moins une solution. À quelle condition cette solution est-elle unique ?

9. Définir une fonction `regression` qui prend en argument un vecteur x et un vecteur y et renvoie le couple w et α solution de (2).

10. Comparer le résultat obtenu à celui donné par la méthode `LinearRegression` du module `sklearn.linear_model`.

Exercise 2 Cas général ; Résolution algébrique

1. *(Génération des données)* On suppose dans un premier temps que la dépendance entre la valeur y_i et les données X_i est linéaire, c'est à dire qu'il existe un vecteur colonne $W \in \mathbb{R}^n$ tel que pour tout index i,

$$y_i = W \cdot X_i$$

   a) Définir une fonction `generationW` qui prend un entier n en argument et génère une matrice W de n lignes et une colonne de valeurs aléatoires dans $[0, 1]$.
   Hint : Utiliser la librairie `numpy` et les fonctions `numpy.random.rand`, `numpy.matrix` et l'opérateur transposé `.T`.
   b) Définir une fonction `generationX` qui prend deux entiers n et N et un réel M en arguments et génère une matrice N lignes et n colonnes tel que pour toute ligne i $X_i \in [-M, M]^n$.
   hint : Utiliser à nouveau `numpy.random.rand` et et `numpy.matrix`.
   c) Calculer le vecteur colonne $y = XW$.

2. *(Détermination des poids W)* On suppose dorénavant qu'on ne connait pas les poids W (on fait "comme si"). Le but est de les retrouver. A cet

effet, on va chercher à minmiser la fonction

$$F(w) = \sum_{i=1}^{N} |w \cdot X_i - y_i|^2.$$

a) Montrer que
$$F(w) = \|Xw - y\|^2.$$
Définir une fonction `Python` qui prend en argument w et calcul $F(w)$. Vérifier que $F(W) = 0$.
Hint : Utiliser `numpy.linalg.norm` pour calculer la norme.

b) Montrer que

$$F(w + \delta w) = F(w) + 2(Xw - y).(X\delta w) + \|X\delta w\|^2.$$

En déduire que w est un minimiseur de F si et seulement si

$$X^T Xw - X^T y = 0$$

Hint : On utilisera que si a et b sont des vecteurs lignes, $\|a + b\|^2 = \|a\|^2 + 2a \cdot b + \|b\|^2$.

c) Montrer que le vecteur poids optimal est défini par

$$w = A^{-1}z$$

où
$$A = X^T X \text{ et } z = X^T y,$$
et A est supposée inversible. Calculer A et z sous `Python` . On utilisera l'opérateur transposé `.T` sur les matrices.

d) Résoudre le sytème linéaire

$$w = A^{-1}z.$$

Vérifier qu'on retrouve bien ainsi les poids W. Hint : Utiliser `numpy.linalg.solve`

Exercise 3 Données bruitées

Modifier le script précédent en ajoutant du bruit dans les données. Plus précisément, on introduira une variable $s = 0.01$ et on ajoutera à chaque valeur y_i un bruit constitué d'un nombre aléatoire dans $[-sM, sM]$.

1. Définir une fonction `generation_y` qui prend en argument M, s, W et X et génère un vecteur y bruité.

2. Calculer l'estimation des poids W à l'aide de ces nouvelles "données" bruitées de 1%.

3. Faire de même avec un bruit de 10% et 50%.

Exercise 4 Gradient réseau de neurone

On peut modéliser la fonction F précédente par un réseau de neurones très simple comportant n nœud en entrée et un unique nœud en sortie. On va définir un tel réseau de neurone, puis définir la fonction coût et enfin appliquer un algorithme d'optimsiation pour retrouver les poids W. On utilisera à cet effet la libraire `Pytorch` .

1. Construire un réseau de neurones `neuralNetwork` ne contenant qu'une couche d'entrée comportant trois nœuds et une couche de sortie avec un nœud unique.
   Hint : Utiliser `nn.Linear` de `Pytorch` .

2. Afficher les poids et le biais du réseau généré.
   Hint : Utiliser `weight` et `bias`.

3. Générer les poids W, les données X et les valeurs y comme lors de l'exercice 2. Convertir au format `FloatTensor` les données X et les valeurs y (on nommera `input` et `target` les nouvelles variables.

4. Calculer la sortie `output` générée par le réseau de neurone initial pour les données X. Convertir les poids w du réseau au format `matrix` de `numpy` et le biais α au format `float`. Vérifier que la sortie du réseau est bien égale à
$$Xw + \alpha.$$

5. Définir la fonction coût F à l'aide de `torch.nn.MSELoss()`

6. Définir la méthode d'optimisation (ici `torch.optim.SGD`) par rapport aux paramètres du réseau. On choisira un taux d'apprentissage (*learning rate*) de 1%.

7. Réaliser une boucle d'optimisation des paramètres du réseau. À chaque itération, on utilisera toutes les données afin d'évaluer le coût.
   Hint : On utilisera les fonctions `zero_grad`, `criterion`, `backward` et `step` de `Pytorch`

8. Vérifier que les poids optimisés sont proches des poids réels et que le biais est proche de zéro.

9. Modifier la boucle d'optimisation pour sauvegarder l'évolution du coût dans un vecteur. Afficher l'évolution du coût à l'aide de la librairie `matplotlib`

10. Quels sont les effets d'une augmentation ou d'une diminution du taux d'apprentissage ?