TP – Exemple de régression multifacteurs

Présentation générale

On a un problème concret pour lequel on souhaite étudier l'évolution d'une grandeur physique en fonction de deux paramètres d'entrées. Pour cela on a collecté un grand nombre de données pour lesquelles on a la valeur de la grandeur physique que l'on souhaite étudier. Pour chacune de ces valeurs (Sorties : y) on a également les valeurs des deux paramètres d'entrées (Entrées1 : x_a et Entrées2 : x_b).

Toutes ces données sont rassemblées dans un fichier texte : « Donnees_mystere.csv ». Ces données sont représentées sur la figure 1.

Le but du TP est d'établir une régression multicritère qui permettrait de prédire la valeur de sortie y_p en fonction des paramètres d'entrée :

$$\mathbf{y_p = f(x_a,x_b)}$$

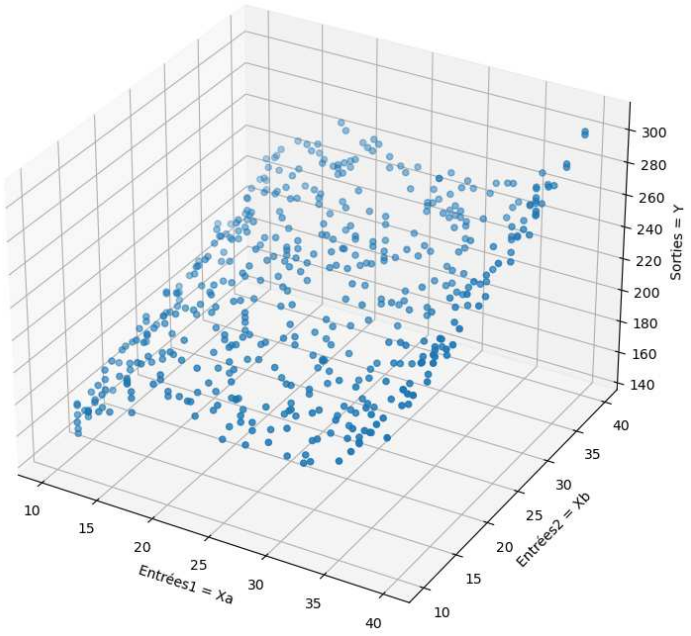

Figure 1 : Représentation graphique 3D des données

1- Analyse des données – Mise en place du modèle de prédiction

On donne en annexe le code qui (Si le fichier « Donnees_mystere.csv » est dans le répertoire courant du Shell) permet de récupérer ces données pour les stocker des les variables **`Entrees`** et **`Sorties`**.

Après exécution du code on lance les commandes suivantes donnant chacune un retour (–>) :

```
type(Entrees) -> <class 'numpy.ndarray'>     et :   np.shape(Entrees)  ->  (500, 2)
type(Sorties) -> <class 'numpy.ndarray'>     et :   np.shape(Sorties)  ->  (500, 1)
```

Question 1 - Donner une description de ces données. (Types et dimensions de **`Entrees`** et **`Sorties`**)

En envisage pour la fonction $y_p = f(x_a,x_b)$ d'établir une régression multifacteurs polynômiale soit une fonction de prédiction du type : $f(x_A,x_B) = \omega_0 + \sum_{i=1}^{n_a} \omega_i.(x_a)^i + \sum_{i=1}^{n_b} \omega_{i+n_a}.(x_b)^i$

Pour cela, on lance la fonction :

```
Trace_donnees(Entrees,
    Sorties, True)
```

donnant le graphique 3D que l'on peut faire tourner pour mieux observer la répartition de ces données.

On obtient notamment les deux vues des figures 2 et 3.

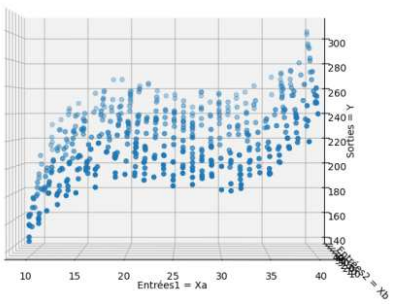

Figure 2 : Répartition x_a , y

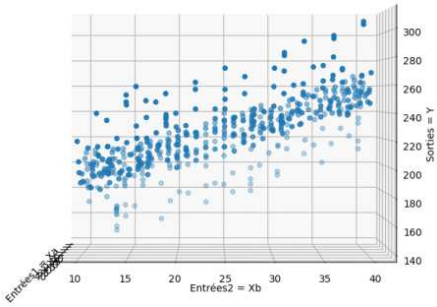

Figure 3 : Répartition x_b , y

Question 2 - Proposer des valeurs cohérentes pour n_a et n_b, les degrés des polynômes de la régression multifacteurs polynômiale : $f(x_a,x_b)$.

2- Préparation des données pour l'algorithme de la descente de gradient

Lire avec attention les lignes de code de la fonction **`Mat_donnees(Entrees,[3,1])`** .

Puis taper les commandes suivantes : **`X = Mat_donnees(Entrees,[3,1])`** et : **`X[0]`**

Cette dernière commande retourne : **`-> array([ 1., 10., 100., 1000., 34.])`**

Question 3 - A quoi correspondent ces cinq valeurs.

Lire avec attention les lignes de code de la fonction **`Init_vecteur_parametres(X,Y)`** qui retourne le vecteur des paramètres (ω_i) initiaux de la régression pour démarrer la descente de gradiant.

Question 4 - Combien a-t-on de paramètres ω_i lorsque **`X = Mat_donnees(Entrees,[3,1])`** ? Comment sont choisis chacun de ces paramètres ?

Lire avec attention les lignes de code de la fonction **`normalisation_donnees(donnees)`** (L'avant dernière ligne de cette fonction est à compléter). Elle prend en argument la matrice des données : d'apprentissage **`donnees`** et retourne un triplet dont le premier élément est la matrice des données normalisées : **`donnees_normal`**. Chaque colonne des données (sauf la première) est translatée et dilatée (multipliée par un réel) de telle sorte que la moyenne des données dans chaque colonne est nulle et l'écart type est égal à 1.

Question 5 - Compléter l'avant dernière ligne de cette fonction de manière à ce que la matrice **`donnees_normal`** soit bien la matrice des données normalisées.

Question 6 - A quoi correspondent les deux autres matrices du triplet renvoyé par la fonction (**`mu`** et **`sigma`**) ? Ces matrices sont des matrices ligne ?

3- Algorithme de la descente de gradient

On dispose des fonctions numpy suivantes qui permettant de faire du calcul vectoriel :

- ☞ **`A.T`** renvoie la transposée de la matrice A.
- ☞ **`A.dot(B)`** renvoie le produit matriciel de A.B
- ☞ **`sum(A)`** renvoie la somme des termes d'une matrice colonne
- ☞ **`np.shape(A)`** renvoie les dimensions de la matrice : Le couple (Nbr lignes, Nbr colonnes)

Question 7 - Implémenter la fonction **`cout(X,W,Y)`**: qui prend en argument les tableaux des données **`X`**, des parametres de la régression **`W`** et des cibles **`Y`** et qui renvoie le cout $J(\theta)$ de la prédiction obtenu avec le vecteur parametre θ = **`W`**. Remarque 2 ou 3 lignes suffisent.

Question 8 - Implémenter la fonction **`gradient(X,W,Y)`**: qui prend en argument les tableaux des données **`X`**, des parametres de la régression **`W`** et des cibles **`Y`** et qui renvoie le vecteur colonne gradient de descente : $\frac{\partial J(\theta)}{\partial \theta}$ avec θ = **`W`** . Remarque 1 ou 2 lignes suffisent.

La fonction suivante **`Regression(Entrees,Sorties,L_degré,Nbr_iter,alpha,Epsilon)`** est la fonction qui va réaliser l'algorithme d'apprentissage par la descente de gradient. C'est le cœur de notre code qui va réaliser l'algorithme donnant les coefficients de la régression linéaire ou polynomiale permettant de faire une prédiction avec les paramètres d'entrée.

Elle prend en argument :

- **`Entrees`** La matrice des données d'apprentissage de dimension : Nbr de données, Nbr de paramètres
- **`Sorties`** La matrice (vecteur) colonne des valeurs cibles (étiquettes des données).
- **`L_degre`** La liste des degrés des polynômes associés aux entrées. Par exemple si on décide de faire une régression polynômiale où le 1ier paramètre (x_A) est de degré 3 et le 2nd de degré 1 Soit pour une fonction de prédiction : $f(x_A,x_B) = \omega_0 + \omega_1.x_A + \omega_2.x_A^2 + \omega_3.x_3^3 + \omega_4.x_B$ on entre la liste d'entiers [3, 1].
- **`Nbr_iter`** Le nombre maximal d'itérations de la descente de gradient à faire pour l'apprentissage.
- **`alpha`** La vitesse d'apprentissage (Learning rate).
- **`Epsilon`** Flottant qui défini la condition d'arrêt de l'apprentissage lorsque le coût n'évolue plus : C'est-à-dire que lorsque la valeur absolue de la différence entre les coûts de deux itérations consécutives seront inférieures à ε l'apprentissage sera stoppé.

Elle ne renvoie rien mais affiche les résultats de l'apprentissage : Affiche dans le Shell les résultats de l'apprentissage (vecteur θ) et affiche dans un graphique en 3D les données d'apprentissage et la surface représentant la fonction de prédiction $f(x_A,x_B) = \omega_0 + \omega_1.x_A + \omega_2.x_A^2 + \omega_3.x_3^3 + \omega_4.x_B$.

Cette fonction procède en six étapes :

- Construction de la matrice **`Xd`** des données à partir de la matrice **`Entrees`** et de la liste **`L_degre`** .
- Normalisation des données **`Xd`** Pour déterminer une matrice des données normalisées **`Xn`**. Et initialisation de $\overset{*}{\theta}$ = **`Wn`** le vecteur colonne des paramètres de la fonction de prédiction sur les données normalisées. Soit le vecteur des paramètres $\overset{*}{\omega}_i$ de la fonction de prédiction : $\overset{*}{f}$(**`Xn`**) = $f(x_A,x_B)$.
- Initialisation du compteur d'itération **`k`** et de la liste historique des coûts **`HC`**. Cette liste mémorise le coût $J(\theta_i)$ à chaque itération. Calcul de la variation du coût (**`Delta_coût`**)
- Boucle d'apprentissage qui prend fin pour k ≥ Nbr_iter ou variation du coût ≤ ε . La 1ière ligne de cette boucle est à compléter. Cet apprentissage permet d'obtenir le vecteur paramètres de la fonction de prédiction (**`Wn`** = $\overset{*}{\theta}$) sur les données normalisées : $\overset{*}{f}$(**`Xn`**) = $f(x_A,x_B)$.
- Détermination des paramètres de la fonction de prédiction sur les données avant normalisation. Les deux lignes de la boucle **`for`** sont à compléter. Elle détermine le vecteur colonne θ = **`W`** paramètres de la fonction de prédiction $f(x_A,x_B)$. Soit par exemple pour **`L_degre = [3,1]`** elle détermine les paramètres ω_i tel que : $f(x_A,x_B) = \omega_0 + \omega_1.x_A + \omega_2.x_A^2 + \omega_3.x_A^3 + \omega_4.x_B$
- Affichage des résultats dans le Shell et sur un graphique 3D.

Question 9 - Compléter la 1ière ligne de la boucle d'apprentissage qui permet d'actualiser à chaque itération de vecteur **`Wn`** des paramètres de la fonction d'apprentissage $\overset{*}{f}$(**`Xn`**) = $f(x_A,x_B)$.

En partant de la manière dont sont normalisées les données, on montre assez facilement qu'on a les relations suivantes entres les paramètres des fonctions de prédiction avant normalisation des données (ω_i) et après normalisation des données ($\overset{*}{\omega}_i$) :

$$\text{Pour : } i \neq 0 : \quad \omega_i = \frac{\overset{*}{\omega}_i}{\sigma_i} \qquad \text{Pour } i = 0 : \quad \omega_0 = \overset{*}{\omega}_0 - \sum_{i=1}^{n_a+n_b} \frac{\overset{*}{\omega}_i . \overline{x_i}}{\sigma_i}$$

où les $\overline{x_i}$ et σ_i sont les moyennes et écarts type des colonnes de la matrice des données **`Xd`**. Ce sont les valeur mémorisées dans les vecteurs ligne **`mu`** et **`sigma`** .

Question 10 - Compléter les deux lignes de la boucle **`for`**.

4- Application de l'algorithme

On souhaite obtenir une fonction de prédiction pour laquelle l'erreur moyenne est inférieure à 10. Soit avoir une fonction de prédiction $f(x_A,x_B)$ telle que : $\varepsilon_{moy} = \sqrt{\frac{1}{m}.\sum_{i=0}^{m-1}\left(f(x_A^i, x_B^i) - y^i\right)^2}$ où x_A^i et x_B^i sont les $i^{ième}$ données d'entrée, y^i la $i^{ième}$ données de sortie et m le nombre de données d'apprentissage.

Question 11 - Taper la commande `Regression(Entrees,Sorties,[1,1],1000,0.5,1e-5)`. Cette commande permet de construire une fonction de prédiction linéaire : $f(x_A,x_B) = \omega_0 + \omega_1.x_A + \omega_2.x_B$. Une régression linéaire simple est-elle suffisante ? Justifier. Etant donné la répartition des données d'apprentissage visible sur le graphique, cela était-il prévisible ?

Question 12 - Taper la commande `Regression(Entrees,Sorties,[3,1],1000,0.5,1e-5)`. Cette commande permet de construire une fonction de prédiction polynomiale pour laquelle le premier paramètre des données est de dégra 3 : $f(x_A,x_B) = \omega_0 + \omega_1.x_A + \omega_2.x_A^2 + \omega_3.x_A^3 + \omega_4.x_B$. Une régression est-elle satisfaisante ? Justifier.

On dispose d'un autre jeu de données d'apprentissage dans le fichier « Donnees_mystere_2.csv ». Avec lequel on souhaite obtenir une fonction de régression polynomiale.

Question 13 - Modifier la ligne 12 du code Python pour charger ces nouvelles données. Puis taper la commande `Trace_donnees(Entrees,Sorties,True)`. Cette commande permet d'afficher ce nouveau jeu de donner. Quels degrés des polynômes des paramètres x_A et x_B proposez vous pour construire une régression ?

Question 14 - Construire cette régression. Conclure.

```
################################################################################
#                     Régression multiple et polynomiale                       #
################################################################################

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

## Importation des données

chemin ="Donnees_mystere.csv"   # Importation des données du fichier ".csv" dans la DataFrame : "dataset"
dataset = pd.read_csv(chemin)
        # Création de la matrice des entrées "X" et du vecteur des sorties "Y"
Entrees = dataset[["Entrées1 = Xa", "Entrées2 = Xb"]].to_numpy(dtype=float)
Sorties = dataset[["Sorties = Y"]].to_numpy(dtype=float

## Affichage des données dans un graphique 3D et Création de la matrice de la régression

def Trace_donnees(X,Y,Montrer):
    ax=plt.axes(projection='3d')
    ax.scatter(X[:,0],X[:,1],Y[:,0])
    ax.set_xlabel('Entrées1 = Xa')
    ax.set_ylabel('Entrées2 = Xb')
    ax.set_zlabel('Sorties = Y')
    if Montrer==True:
        plt.show()

def Mat_donnees(entrees_brutes:np.array,L_degres:list)->np.array:
    Ndonnees,Nparam=np.shape(entrees_brutes)
    donnees=np.ones((Ndonnees,1+sum(L_degres)))
    colonne=1
    for ip in range(Nparam):
        degre=1
        for id in range(L_degres[ip]):
            donnees[:,colonne]=(entrees_brutes[:,ip])**degre
            degre+=1
            colonne+=1
    return donnees
```

```
def Init_vecteur_parametres(X,Y):
    Nbr_para=np.shape(X)[1]
    Winit=[[np.mean(Y)]]
    Winit=Winit+[[rd.random()] for i in range(1,Nbr_para)]
    return np.array(Winit)

def normalisation_donnes(donnees):
    mu = np.ones(donnees.shape[1])          #  Moyennes des entrées
    sigma = np.zeros(donnees.shape[1])      # Ecart type des entrées
    donnees_normal=np.ones(donnees.shape)   # Matrice des entrées normalisées
    for k in range(1,donnees.shape[1]):     # Pour chaque colonne sauf celle des "1"
        mu[k] = np.mean(donnees[:,k])                   # Calcul de la moyenne
        sigma[k] = np.std(donnees[:,k])                 # Calcul de l'écart type
                                                        # Normalisation des données                # A compléter
    return donnees_normal,mu,sigma


## Algorithme de la régression

def Cout(X,W,Y):

                                                                                                   # A compléter



def gradient(X,W,Y):
                                                                                                   # A compléter
```

```
def Regression(X,Sorties,L_degre,Nbr_iter,alpha,Epsilon):
        # Construction de la matrice de données
    Xd=Mat_donnees(Entrees,L_degre)
        # Normalisation des données
    Xn,mu,sigma=normalisation_donnes(Xd)
    Wn=Init_vecteur_parametres(Xn,Sorties)
    print('Parametres initiaux de la régression :')
    print(Wn)
        # Initialisation du Delta_coût et du compteur
    cout=Cout(Xn,Wn,Sorties)
    HC=[cout]                   # Historique des coûts
    CP=np.inf                   # Coût précédent (infini)
    k=0                         # Compteur d'itérations
    Delta_Cout=abs(CP-HC[-1])   # Variation du coût
    print(cout,'\t',k,'\t',Delta_Cout)
        # Boucle d'apprentissage par la descente de gradiant
    while (k<Nbr_iter and Delta_Cout>Epsilon):
                                            # Actualisation du vecteur paramètre Wn          # A compléter
            # Calcul du nouveau Delta_coût
        CP=HC[-1]
        cout=Cout(Xn,Wn,Sorties)
        HC.append(cout)
        Delta_Cout=abs(CP-HC[-1])
            # Affichage du Coût, du compteur et du Delta_coüt
            #  toutes les 1000 itérations
        k+=1    # Incrémentation du compteur d'itération
        if k%1000==0:
            print(cout,'\t',k,'\t',Delta_Cout)
                # Le vecteur paramètres est celui des données normalisées
                # Caclul des paramètres des données avant normalisation
    W=np.zeros((len(Wn),1))
    W[0,0]=Wn[0,0]
    for i in range(1,len(Wn)):
                                        # Pour le calcul du paramètre Omega 0            # A compléter
                                        # Pour le calcul du paramètre Omega i
            # Affiche graphique des données et de la régression
    Ecriture_regression(W,k,L_degre,HC)
    graphique(Entrees,Sorties,W,L_degre)
    # return W,HC,k
```

```
## Affichage des résultats et des données dans un graphique 3D

def Ecriture_regression(W,k,L_degre,Liste_Couts):
    print("La régression a été construite en",k,"itérations")
    print("Le modèle de prédiction est :")
    equation='y = w0 + w1.xa'
    for j in range(2,L_degre[0]+1):
        equation +=' + w'+str(j)+'.xa^'+str(j)
    equation+=' + w'+str(L_degre[0]+1)+'.xb'
    for j in range(2,L_degre[1]+1):
        equation +=' + w'+str(L_degre[0]+2)+'.xb^'+str(j)
    print(equation)
    print('Les paramètres wi de cette équation sont (de w0 à w'+str(sum(L_degre))+') :')
    print(W)
    print("L'erreur moyenne est :",np.sqrt(Liste_Couts[-1])*2)

def graphique(X,Y,W,L_degres):
    Trace_donnees(X,Y,False)
    ax=plt.axes(projection='3d')
    ax.scatter(X[:,0],X[:,1],Y[:,0])
    ax.set_xlabel('Entrées1 = Xa')
    ax.set_ylabel('Entrées2 = Xb')
    ax.set_zlabel('Sorties = Y')
    f = lambda sx, sy : W[0,0] + sum([W[i+1,0]*sx**(i+1) for i in range(L_degres[0])]) +
sum([W[i+1+L_degres[0],0]*sy**(i+1) for i in range(L_degres[1])])
    GX = np.linspace(10,40,31)
    GY = np.linspace(10,40,31)
    GX, GY = np.meshgrid(GX, GY)
    Z = f(GX, GY)
    ax.plot_surface(GX, GY, Z, cmap='rainbow', alpha=0.8)
    plt.show()
```