td_note_2017_2

July 1, 2022

1 1A.e - TD noté, 21 février 2017

Solution du TD noté, celui-ci présente un algorithme pour calculer les coefficients d'une régression quantile et par extension d'une médiane dans un espace à plusieurs dimensions.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

Précision : dans tout l'énoncé, la transposée d'une matrice est notée $X' = X^T$. La plupart du temps X et Y désignent des vecteurs colonnes. β désigne un vecteur ligne, W une matrice diagonale.

2 Exercice 1

2.0.1 Q1

A l'aide du module random, générer un ensemble de points aléatoires.

[2]:
```
import random

def ensemble_aleatoire(n):
    res = [random.randint(0, 100) for i in range(n)]
    res[0] = 1000
    return res

ens = ensemble_aleatoire(10)
ens
```

[2]:
```
[1000, 51, 83, 29, 15, 62, 90, 28, 61, 40]
```

2.0.2 Q2

La médiane d'un ensemble de points $\{X_1, ..., X_n\}$ est une valeur X_M telle que :

$$\sum_i \mathbb{1}_{X_i<X_m} = \sum_i \mathbb{1}_{X_i>X_m}$$

Autrement dit, il y a autant de valeurs inférieures que supérieures à X_M. On obtient cette valeur en triant les éléments par ordre croissant et en prenant celui du milieu.

[3]:
```
def mediane(ensemble):
    tri = list(sorted(ensemble))
    return tri[len(tri)//2]
```

```
mediane(ens)
```

[3]: 61

2.0.3 Q3

Lorsque le nombre de points est pair, la médiane peut être n'importe quelle valeur dans un intervalle. Modifier votre fonction de façon à ce que la fonction précédente retourne le milieu de la fonction.

[4]:
```
def mediane(ensemble):
    tri = list(sorted(ensemble))
    if len(tri) % 2 == 0:
        m = len(tri)//2
        return (tri[m] + tri[m-1]) / 2
    else:
        return tri[len(tri)//2]

mediane(ens)
```

[4]: 56.0

2.0.4 Q4

Pour un ensemble de points $E = \{X_1, ..., X_n\}$, on considère la fonction suivante :

$$f(x) = \sum_{i=1}^{n} |x - X_i|$$

.

On suppose que la médiane X_M de l'ensemble E n'appartient pas à E : $X_M \notin E$. Que vaut $f'(X_M)$? On acceptera le fait que la médiane est le seul point dans ce cas.

$$f'(X_m) = -\sum_{i=1}^{n} \mathbb{1}_{X_i < X_m} + \sum_{i=1}^{n} \mathbb{1}_{X_i > X_m}$$

Par définition de la médiane, $f'(X_M) = 0$. En triant les éléments, on montre que la $f'(x) = 0 \iff x = X_m$.

2.0.5 Q5

On suppose qu'on dispose d'un ensemble d'observations (X_i, Y_i) avec $X_i, Y_i \in \mathbb{R}$. La régression linéaire consiste une relation linéaire $Y_i = aX_i + b + \epsilon_i$ qui minimise la variance du bruit. On pose :

$$E(a, b) = \sum_i (Y_i - (aX_i + b))^2$$

On cherche a, b tels que :

$$a^*, b^* = \arg\min E(a, b) = \arg\min \sum_i (Y_i - (aX_i + b))^2$$

La fonction est dérivable et on trouve :

$$\frac{\partial E(a,b)}{\partial a} = -2\sum_i X_i(Y_i - (aX_i + b)) \text{ et } \frac{\partial E(a,b)}{\partial b} = -2\sum_i (Y_i - (aX_i + b))$$

Il suffit alors d'annuler les dérivées. On résoud un système d'équations linéaires. On note :

$$\mathbb{E}X = \frac{1}{n}\sum_{i=1}^{n} X_i \text{ et } \mathbb{E}Y = \frac{1}{n}\sum_{i=1}^{n} Y_i$$
$$\mathbb{E}X^2 = \frac{1}{n}\sum_{i=1}^{n} X_i^2 \text{ et } \mathbb{E}XY = \frac{1}{n}\sum_{i=1}^{n} X_i Y_i$$

Finalement :

$$a^* = \frac{\mathbb{E}XY - \mathbb{E}X\mathbb{E}Y}{\mathbb{E}X^2 - (\mathbb{E}X)^2} \text{ et } b^* = \mathbb{E}Y - a^*\mathbb{E}X$$

Lorsqu'on a plusieurs dimensions pour X, on écrit le problème d'optimisation, on cherche les coefficients β^* qui minimisent :

$$E(\beta) = \sum_{i=1}^{n} (y_i - X_i\beta)^2 = \|Y - X\beta\|^2$$

La solution est : $\beta^* = (X'X)^{-1}X'Y$.
Ecrire une fonction qui calcule ce vecteur optimal.

[5]:
```
from numpy.linalg import inv

def regression_lineaire(X, Y):
    t = X.T
    return inv(t @ X) @ t @ Y

import numpy
X = numpy.array(ens).reshape((len(ens), 1))
regression_lineaire(X, X+1)   # un essai pour vérifier que la valeur n'est pas␣
 ↪aberrante
```

[5]:
```
array([[ 1.00141843]])
```

2.0.6 Q6

Ecrire une fonction qui transforme un vecteur en une matrice diagonale.

[6]:
```
def matrice_diagonale(W):
    return numpy.diag(W)

matrice_diagonale([1, 2, 3])
```

[6]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

2.0.7 Q7

On considère maintenant que chaque observation est pondérée par un poids w_i. On veut maintenant trouver le vecteur β qui minimise :

$$E(\beta) = \sum_{i=1}^{n} w_i (y_i - X_i\beta)^2 = \left\|W^{\frac{1}{2}}(Y - X\beta)\right\|^2$$

Où $W = diag(w_1, ..., w_n)$ est la matrice diagonale. La solution est :

$$\beta_* = (X'WX)^{-1}X'WY$$

.
Ecrire une fonction qui calcule la solution de la régression pondérée. La fonction ravel est utile.

[7]:
```
def regression_lineaire_ponderee(X, Y, W):
    if len(W.shape) == 1 or W.shape[0] != W.shape[1]:
        # c'est un vecteur
        W = matrice_diagonale(W.ravel())
    wx = W @ X
    xt = X.T
    return inv(xt @ wx) @ xt @ W @ Y


X = numpy.array(sorted(ens)).reshape((len(ens), 1))
Y = X.copy()
Y[0] = max(X)
W = numpy.ones(len(ens))
W[0] = 0
regression_lineaire_ponderee(X, Y, W), regression_lineaire(X, Y)
```

[7]:
```
(array([[ 1.]]), array([[ 1.01240451]]))
```

2.0.8 Q8

Ecrire une fonction qui calcule les quantités suivantes (fonctions maximum, reciprocal).

$$z_i = \frac{1}{\max(\delta, |y_i - X_i\beta|)}$$

[8]:
```
def calcule_z(X, beta, Y, W, delta=0.0001):
    epsilon = numpy.abs(Y - X @ beta)
    return numpy.reciprocal(numpy.maximum(epsilon, numpy.ones(epsilon.shape) * delta))

calcule_z(X * 1.0, numpy.array([[1.01]]), Y, W)
```

[8]:
```
array([[  1.01330469e-03],
       [  5.26315789e+00],
       [  4.54545455e+00],
       [  3.22580645e+00],
       [  1.85185185e+00],
       [  1.47058824e+00],
       [  1.14942529e+00],
       [  1.07526882e+00],
       [  1.07526882e+00],
       [  1.00000000e-01]])
```

2.0.9 Q9

On souhaite coder l'algorithme suivant :

1. $w_i^{(1)} = 1$
2. $\beta_{(t)} = (X'W^{(t)}X)^{-1}X'W^{(t)}Y$
3. $w_i^{(t+1)} = \frac{1}{\max(\delta, |y_i - X_i\beta^{(t)}|)}$
4. $t = t + 1$
5. Retour à l'étape 2.

[9]:
```
def algorithm(X, Y, delta=0.0001):
    W = numpy.ones(X.shape[0])
    for i in range(0, 10):
        beta = regression_lineaire_ponderee(X, Y, W)
        W = calcule_z(X, beta, Y, W, delta=delta)
        E = numpy.abs(Y - X @ beta).sum()
        print(i, E, beta)
    return beta

X = numpy.random.rand(10, 1)
Y = X*2 + numpy.random.rand()
Y[0] = Y[0] + 100
algorithm(X, Y)
```

```
0 150.13052808 [[ 13.82243581]]
1 104.79608014 [[ 3.21524459]]
2 100.851019446 [[ 2.25815451]]
3 100.36420567 [[ 2.12644545]]
4 100.255554539 [[ 2.09141327]]
5 100.220626093 [[ 2.0777948]]
6 100.219023635 [[ 2.07639404]]
7 100.21901041 [[ 2.07631459]]
8 100.218994922 [[ 2.07622156]]
9 100.218976948 [[ 2.07611358]]
```

[9]:
```
array([[ 2.07611358]])
```

[10]:
```
regression_lineaire(X, Y)
```

[10]:
```
array([[ 13.82243581]])
```

2.0.10 Q10

[11]:
```
ens = ensemble_aleatoire(10)
Y = numpy.empty((len(ens), 1))
Y[:,0] = ens
X = numpy.ones((len(ens), 1))
mediane(ens)
```

[11]:
```
34.5
```

[12]:
```
Y.mean(axis=0)
```

[12]:
```
array([ 131.1])
```

[13]:
```
regression_lineaire(X, Y)
```

[13]:
```
array([[ 131.1]])
```

[14]:
```
algorithm(X,Y)
```

```
0 1737.8 [[ 131.1]]
1 1215.2110733 [[ 55.05276833]]
2 1196.55478823 [[ 48.77739411]]
3 1190.4919578 [[ 45.7459789]]
4 1183.56462833 [[ 42.28231416]]
5 1179.0 [[ 39.7931558]]
6 1179.0 [[ 39.7931558]]
7 1179.0 [[ 39.7931558]]
8 1179.0 [[ 39.7931558]]
9 1179.0 [[ 39.7931558]]
```

[14]:
```
array([[ 39.7931558]])
```

[15]:
```
mediane(ens)
```

[15]:
```
34.5
```

[16]:
```
list(sorted(ens))
```

[16]:
```
[5, 6, 12, 14, 29, 40, 52, 67, 86, 1000]
```

La régression linéaire égale la moyenne, l'algorithme s'approche de la médiane.

2.1 Quelques explications et démonstrations

Cet énoncé est inspiré de Iteratively reweighted least squares. Cet algorithme permet notamment d'étendre la notion de médiane à des espaces vectoriels de plusieurs dimensions. On peut détermine un point X_M qui minimise la quantité :

$$\sum_{i=1}^{n} |X_i - X_M|$$

Nous reprenons l'algorithme décrit ci-dessus :

1. $w_i^{(1)} = 1$
2. $\beta_{(t)} = (X'W^{(t)}X)^{-1}X'W^{(t)}Y$
3. $w_i^{(t+1)} = \frac{1}{\max(\delta, |y_i - X_i\beta^{(t)}|)}$
4. $t = t + 1$
5. Retour à l'étape 2.

L'erreur quadratique pondéré est :

$$E_2(\beta, W) = \sum_{i=1}^{n} w_i \left\| Y_i - X_i\beta \right\|^2$$

Si $w_i = \frac{1}{|y_i - X_i\beta|}$, on remarque que :

$$E_2(\beta, W) = \sum_{i=1}^{n} \frac{\left\| Y_i - X_i\beta \right\|^2}{|y_i - X_i\beta|} = \sum_{i=1}^{n} |y_i - X_i\beta| = E_1(\beta)$$

On retombe sur l'erreur en valeur absolue optimisée par la régression quantile. Comme l'étape 2 consiste à trouver les coefficients β qui minimise $E_2(\beta, W^{(t)})$, par construction, il ressort que :

$$E_1(\beta^{(t+1)}) = E_2(\beta^{(t+1)}, W^{(t)}) \leqslant E_2(\beta^{(t)}, W^{(t)}) = E_1(\beta^{(t)})$$

La suite $t \rightarrow E_1(\beta^{(t)})$ est suite décroissant est minorée par 0. Elle converge donc vers un minimum. Or la fonction $\beta \rightarrow E_1(\beta)$ est une fonction convexe. Elle n'admet qu'un seul minimum (mais pas nécessaire un seul point atteignant ce minimum). L'algorithme converge donc vers la médiane. Le paramètre δ est là pour éviter les erreurs de divisions par zéros et les approximations de calcul faites par l'ordinateur.

2.2 Quelques commentaires sur le code

Le symbol [@](https://www.python.org/dev/peps/pep-0465/) a été introduit par Python 3.5 et est équivalent à la fonction numpy.dot. Les dimensions des matrices posent souvent quelques problèmes.

[17]:
```
import numpy
y = numpy.array([1, 2, 3])
M = numpy.array([[3, 4], [6, 7], [3, 3]])
M.shape, y.shape
```

[17]:
```
((3, 2), (3,))
```

[18]:
```
try:
     M @ y
except Exception as e:
    print(e)
```

```
shapes (3,2) and (3,) not aligned: 2 (dim 1) != 3 (dim 0)
```

Par défaut, numpy considère un vecteur de taille `(3,)` comme un vecteur ligne `(3,1)`. Donc l'expression suivante va marcher :

[19]:
```
y @ M
```

[19]:
```
array([24, 27])
```

Ou :

[20]:
```
M.T @ y
```

[20]:
```
array([24, 27])
```

[21]:
```

```