TD : concevoir un programme en Python

Alexandre Roulois (Université Paris Cité, LLF, CNRS)

Après la courte introduction aux bonnes pratiques en matière de structuration d'un programme en Python, vous allez écrire votre premier réel programme.

Rappel sur l'exécution d'un script

Téléchargez le script *get_hypotenuse.py* sur votre machine à un endroit facile d'accès. Exécutez-le ensuite depuis une ILC (fenêtre de terminal) :

```
$ python /path/to/get_hypotenuse.py
```

Si une erreur apparaît, lisez le message, il vous indique sans doute un problème dans l'expression du chemin vers le fichier.

Effectuer une régression linéaire

Pour la petite histoire, c'est en 1886 que le terme de régression apparaît pour la première fois, dans un article de Sir Francis Galton qu'il publie sous le titre de *Regression Towards Mediocrity in Hereditary Stature*. Dans cet article, il mettait en évidence que les enfants de personnes de grandes tailles avaient tendance à être plus petits qu'elles, et inversement, d'où l'idée d'une régression vers la médiocrité pour décrire en fait un phénomène d'attraction de la moyenne.

À partir de données simulées, votre objectif sera de calculer la droite de régression avec la méthode des moindres carrés. Vous appliquerez les notions abordées dans le précédent TD afin de réaliser votre programme.

Dans votre répertoire de travail, créez un nouveau fichier Python que vous intitulerez *linear-regression.py*. Préparez le squelette de votre script.

Charger les modules nécessaires

Chargez d'abord, à l'endroit prévu à cet effet, toutes les librairies qui seront utilisées par votre programme :

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

Générer des données aléatoires

Vous disposez de la fonction ci-dessous qui génère des données à l'aspect linéaire. Intégrez-la à votre programme.

```
def generate_data():
    """Generate random data from a linear function."""
    noise = np.random.uniform(low=-20, high=20, size=(100,))
    x = np.arange(stop=100)
    y = 2 * x - 4 + noise

    return (x, y)
```

Dans la procédure principale, récupérez les données générées de manière aléatoire et affichez-les dans un diagramme :

```
# get data
X, Y = generate_data()

# make plot
coef = np.polyfit(X, Y, 1)
poly1d_fn = np.poly1d(coef)
plt.plot(X, Y, 'yo', X, poly1d_fn(X), '--k')

# save plot
plt.savefig('points.png', dpi=72)
```

Exécutez le script depuis une fenêtre de terminal avec l'utilitaire `python`. Si un fichier *points.png* est apparu dans l'interface, vous pouvez l'ouvrir afin d'afficher le résultat de vos manipulations et passer à l'étape suivante ; autrement, recommencez depuis le début.

Une droite de régression des moindres carrés

Lorsque nous prenons les coordonnées d'une observation, nous remarquons qu'elles sont éloignées de la droite. Il existe un décalage – que l'on appelle communément une erreur – et la droite de régression des moindres carrés est celle qui minimise la somme des carrés de toutes les erreurs. Le rapport s'établit au carré afin d'éviter les valeurs négatives.

La formule

L'équation réduite qui permet d'obtenir les coordonnées de tous les points de la droite et, partant, d'obtenir une prédiction de $\hat{y}$ en fonction de x respecte la forme $\hat{y} = mx + b$. Deux étapes majeures pour la trouver, calculer d'abord le coefficient directeur m puis l'ordonnée à l'origine b.

Calculer le coefficient directeur

La résolution du coefficient directeur d'une droite des moindres carré est régi par la formule ci-dessous :

$$m = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - \left(\sum x\right)^2}$$

De cette formule, nous concluons avoir besoin de connaître :

- le nombre n des observations ;
- la somme du produit de x et de y ;
- la somme de x et son carré ;
- la somme de y et la somme de x ;
- la somme des carrés de x.

Ajoutez à votre programme la fonction ci-dessous qui se charge des calculs :

```
def slope(x, y):
    """Return the slope of a straight line,
    with the least squares method.

    Arguments:
    x -- data in x-axis
    y -- data in y-axis
    """

    n = len(x)
```

```
    sum_xy = sum(x * y)
    sum_x_squared = sum(x) ** 2
    sum_x = sum(x)
    sum_y = sum(y)
    sum_squares_x = sum(x ** 2)

    # formula
    m = (n * sum_xy - sum_x * sum_y) / (n * sum_squares_x - sum_x_squared)

    return m
```

Calculer l'ordonnée à l'origine

La formule de résolution de l'ordonnée à l'origine fait appel au coefficient directeur et aux moyennes des valeurs de x et de y :

$$b = \bar{y} - m\bar{x}$$

```
def intercept(m, x, y):
    """Intercept of a straight line.
    Arguments:
    m -- slope
    x -- data in x-axis
    y -- data in y-axis
    """

    avg_x = sum(x) / len(x)
    avg_y = sum(y) / len(y)
    b = avg_y - (m * avg_x)

    return b
```

Vérification

La droite qui minimise la somme des carrés des erreurs s'établit tout simplement grâce à l'équation réduite de toute droite, à l'exception ici que vous effectuerez des prédictions :

$$\hat{y} = b + mx$$

Ajoutez une fonction qui calcule la coordonnée prédite en y d'un point relativement à un x connu :

```
def F(*, x, m, b) -> float:
    """Solution to the standard form equation
    of a straight line.

    Keyword-only arguments:
    x -- value of x
    m -- slope
    b -- intercept
    """
    return m * x + b
```

Dans la procédure principale, ne conservez que la toute première instruction qui génère les données et supprimez tout le reste. À partir des données, faites appel aux fonction `slope()` et `intercept()` définies précédemment pour récupérer le coefficient directeur et l'ordonnée à l'origine de la droite de régression des moindres carrés :

```
# slope & intercept of a straight line
m = slope(X, Y)
b = intercept(m, X, Y)
```

Ensuite, il ne vous reste plus qu'à effectuer des prédictions pour chaque valeur de x avec la fonction `F()` :

```
Y_pred = [ F(x=x, m=m, b=b) for x in X ]
```

Toujours dans la procédure principale, sauvegardez maintenant le graphique :

```
ax = plt.subplots()

ax = plt.plot(X, Y_pred)
ax = plt.scatter(X, Y)

plt.savefig('linear-regression.png', dpi=72)
```

Exécutez votre script. Si un fichier *linear-regression.png* apparaît, vous pouvez tenter le défi suivant : serez-vous capable d'encapsuler la procédure principale dans une fonction `main()` ?