Python pour les data scientistes

Chapitre 7 : Le package SciPy

1. Introduction

SciPy est une librairie de calcul scientifique qui utilise NumPy en dessous. SciPy est l'abréviation de Scientific Python.
Elle fournit davantage de fonctions utilitaires pour l'optimisation, les statistiques et le traitement du signal. Comme NumPy, SciPy est open source et peut donc être utilisé librement.
SciPy a été créé par le créateur de NumPy, Travis Olliphant.
Si SciPy utilise NumPy à la base, pourquoi ne pouvons-nous pas utiliser NumPy ?
SciPy a optimisé et ajouté des fonctions qui sont fréquemment utilisées dans NumPy et la science des données.

2. Installation de SciPy

Si Python et PIP sont déjà installés sur votre système, l'installation de SciPy est très simple.

Pour l'installer, utiliser la commande : **pip install scipy**

Une fois SciPy installé, importez le ou les modules SciPy que vous souhaitez utiliser dans vos applications en ajoutant l'instruction : **from scipy import module**

3. Constantes dans SciPy

Comme SciPy est davantage axé sur les implémentations scientifiques, il fournit de nombreuses constantes scientifiques intégrées. Ces constantes peuvent être utiles lorsque vous travaillez dans le domaine de la science des données.

PI est un exemple de constante scientifique.

Exemple : Combien de mètres cubes y a-t-il dans un litre ?

```
from scipy import constants
print(constantes.liter)
```

```
0.001
```

SciPy propose un ensemble de constantes mathématiques, l'une d'entre elles est liter qui renvoie 1 litre sous forme de mètres cubes.

Une liste de toutes les unités sous le module constantes peut être vue en utilisant la fonction dir().

Exemple : Liste de toutes les constantes

```
from scipy import constants
print(dir(constants))
```

4. Optimiseurs dans SciPy

Les optimiseurs sont un ensemble de procédures définies dans SciPy qui trouvent soit la valeur minimale d'une fonction, soit la racine d'une équation.
Essentiellement, tous les algorithmes d'apprentissage automatique ne sont rien d'autre qu'une équation complexe qui doit être minimisée à l'aide de données.

5. Données éparses SciPy

Les données éparses sont des données dont la plupart des éléments sont inutilisés (éléments qui ne portent aucune information).

Il peut s'agir d'un tableau comme celui-ci :

[1, 0, 2, 0, 0, 3, 0, 0, 0, 0, 0, 0]

Données éparses : c'est un ensemble de données où la plupart des valeurs des éléments sont nulles.

Tableau dense : est l'opposé d'un tableau clairsemé : la plupart des valeurs ne sont pas nulles.

SciPy possède un module, **scipy.sparse**, qui fournit des fonctions permettant de traiter les données éparses.

Nous utilisons principalement deux types de matrices éparses :

- CSC - Compressed Sparse Column. Pour une arithmétique efficace, un découpage rapide des colonnes.

- CSR - Compressed Sparse Row. Pour un découpage rapide des lignes, des produits vectoriels de matrice plus rapides.

Exemple : Créer une matrice CSR à partir d'un tableau :

```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([0, 0, 0, 0, 0, 1, 1, 0, 2])
print(csr_matrix(arr))
```

```
  (0, 5)        1
  (0, 6)        1
  (0, 8)        2
```

D'après le résultat, nous pouvons voir qu'il y a 3 éléments avec une valeur.
L'élément 1 est dans la rangée 0, position 5 et a la valeur 1.
L'élément 2 se trouve à la ligne 0, position 6 et a la valeur 1.
L'élément 3 se trouve à la ligne 0, position 8 et a la valeur 2.

Méthodes pour les matrices éparses
Visualisation des données stockées (pas les éléments zéro) avec la propriété data :

```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])
print(csr_matrix(arr).data)
```

```
[1 1 2]
```

Exemple : Compter les nonzéros avec la méthode count_nonzero() :

```
import numpy as np
from scipy.sparse import csr_matrix
```

```
arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])
print(csr_matrix(arr).count_nonzero())
```

```
3
```

Exemple : Suppression des entrées zéro de la matrice avec la méthode eliminate_zeros() :

```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])
mat = csr_matrix(arr)
mat.eliminate_zeros()
print(mat)
```

Exemple : Élimination des entrées en double avec la méthode **sum_duplicates()**

```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])
mat = csr_matrix(arr)
mat.sum_duplicates()
print(mat)
```

```
  (1, 2)        1
  (2, 0)        1
  (2, 2)        2
```

6. Graphiques SciPy

Les graphes sont une structure de données essentielle.
SciPy nous fournit le module scipy.sparse.csgraph pour travailler avec de telles structures de données.
Utilisez la méthode de Dijkstra pour trouver le plus court chemin dans un graphe d'un élément à un autre.

Elle prend les arguments suivants :

1. return_predecessors : booléen (True pour retourner le chemin entier de la traversée, sinon False).

2. indices : indice de l'élément pour retourner tous les chemins à partir de cet élément seulement.

3. limit : poids maximal du chemin.

Exemple : Trouvez le plus court chemin de l'élément 1 à l'élément 2

```
import numpy as np
from scipy.sparse.csgraph import dijkstra
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 2],
  [1, 0, 0],
  [2, 0, 0]
])
newarr = csr_matrix(arr)
print(dijkstra(newarr, return_predecessors=True, indices=0))
```

```
(array([0., 1., 2.]), array([-9999,     0,     0], dtype=int32))
```

- Profondeur d'abord

La méthode depth_first_order() renvoie une traversée en profondeur d'un nœud.

Cette fonction prend les arguments suivants :

le graphe.

l'élément de départ pour traverser le graphe.

```
import numpy as np
from scipy.sparse.csgraph import depth_first_order
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 0, 1],
  [1, 1, 1, 1],
  [2, 1, 1, 0],
  [0, 1, 0, 1]
])
newarr = csr_matrix(arr)
print(depth_first_order(newarr, 1))
```

- Largeur d'abord

La méthode breadth_first_order() renvoie une traversée de type breadth first order à partir d'un nœud.

Cette fonction prend les arguments suivants :

le graphe.

l'élément de départ pour traverser le graphe.

Exemple : Traverse avec la largeur d'abord d'une une matrice d'adjacence donnée :

```
import numpy as np
from scipy.sparse.csgraph import breadth_first_order
from scipy.sparse import csr_matrix
arr = np.array([
  [0, 1, 0, 1],
  [1, 1, 1, 1],
  [2, 1, 1, 0],
  [0, 1, 0, 1]
])
newarr = csr_matrix(arr)
print(breadth_first_order(newarr, 1))
```