Université de Rouen
Master 2 GIL
Matière : Machine Learning en Python

TP 2
Numpy - SciPy

Les classes et fonctions de cette séance doivent être définies dans un module `tp2`.

Question 1.- Définir une fonction `exo1()` permettant de :

1. Créer un tableau ndarray 4×5 de réels sur 4 octets valant tous 1.
2. Créer un tableau 4×5 de réels sur 4 octets avec quatres lignes dont les valeurs sont 1, 2, 3, 4, 5.
3. Concaténer horizontalement, puis verticalement, les deux derniers tableaux. .
4. Créer un tableau 4×5 de valeurs aléatoires suivant la loi normale avec une moyenne de 0 et un écart type de 3.
5. Afficher la dernière ligne, puis la dernière colonne de ce dernier tableau.
6. Redimensionner ce tableau en 5×4.
7. Créer à partir de tableau, un sous-tableau qui ne contient pas la première ligne, ni la première colonne.
8. Créer un tableau à une dimension qui contient les valeurs positives de ce tableau.
9. Créer un tableau qui contient les valeurs positives du dernier tableau et 0 si les valeurs sont négatives.

Question 2.- On veut créer une classe `NDArray` qui hérite de `numpy.ndarray` et posséde des fonctionnalités supplémentaires.

1. Définir la `NDArray` qui hérite de `numpy.ndarray`. Comme `numpy.ndarray` est écrite en C il faut redéfinir la méthode `__new__` et pas `__init__`, voir ce lien.
2. **Remplissage des valeurs manquantes (imputation).** Les valeurs manquantes (non définies ou infinies) peuvent perturber l'analyse des données. Une solution consiste à les remplacer par des valeurs comme la moyenne ou la médiane des données disponibles. Définir la méthode `filled(self, strategy="mean")` qui retourne le tableau avec remplacement des valeurs manquantes par la moyenne ou la médiane.
3. **Normalisation des données**. La normalisation des données consiste à mettre toutes les valeurs dans une plage donnée, souvent entre 0 et 1. La formule pour normaliser un vecteur x est :

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Définir la méthode `normalized(self)` qui retourne le tableau normalisé.

4. **La normalisation standard**, également appelée standardisation ou normalisation z-score, consiste à soustraire la moyenne et à la diviser par l'écart type. Dans ce cas, chaque valeur refléterait la distance par rapport à la moyenne en unités d'écart-type. La formule pour normaliser un vecteur x en utilisant sa moyenne μ et son écart-type σ est :

   $$x_{\text{norm}} = \frac{x - \mu}{\sigma}$$

   Cette méthode permet d'obtenir des valeurs centrées autour de 0, avec un écart-type de 1. Définir la méthode `std_normalized(self)` qui retourne le tableau normalisé de cette façon.
5. **Test de Student.** Ajouter une méthode `ttest(self, other)` qui retourne le résultat du test de Student de `self` avec `other`.
6. Surcharger les opérateurs arithmétiques pour pourvoir faire des opérations entre `NDArray` et une matrice Compressed Sparse Row (CSR). Le résultat doit être une CSR. Indication : si on ajoute un `numpy.ndarray` à une CSR on obtient une `numpy.matrix` qui n'est pas compressée.
7. On voudrais garder les informations sur la provenance d'un tableau `NDArray` quand il est créé par une opération d'indexation. On veut garder sous la forme de propriétés le tableau `base`, et un tableau `indexed` de booléens avec des valeurs à `True` si l'indice correspondant a été indexé dans l'opération. Par exemple si on fait :

   ```
   a = NDArray(np.arange(6).reshape(2, 3) + 11)
   b = a[1:, 1:]
   print("b = a[1:, 1:] =", b)
   print("b.base", b.base)
   print("b.masked", b.indexed)
   ```

   ça affichera :

   ```
   b = a[1:, 1:] = [[15 16]]
   b.base
    [[11 12 13]
    [14 15 16]]
   b.indexed
    [[ False  False  False]
    [ False True True]]
   ```

Définir les propriétés `base` et `indexed` associées aux attributs `_base` et `_indexed`. Pour celà on redéfinit `__getitem__(self, item)` comme ceci :

(a) on appelle `obj = super().__getitem__(item)`,
(b) on transforme `obj` en type `NDArray`
(c) on rajoute `obj._base = self`
(d) et on retourne `obj`

On fait pareillement pour créér `obj._indexed` avant de retourner `obj`.

Remarques :

(a) `obj._indexed[item] = True` va déclencher une exception dans un appel interne de Numpy, il faut juste la mettre dans un `try... except pass`.
(b) Initialement (dans `__new__`) on initialise `_base` et `_indexed` à `None`