Cose utili per iniziare

Questo non è un corso di Python, ma raccogliamo comunque alcune funzioni utili per quello che dobbiamo fare.

È utile la documentazione che si trova qui: http://python.it/doc/Howtothink/Howtothink-html-it/index.htm

0.1 Liste

Per prima cosa impariamo a creare una lista. In python è possibile creare liste di elementi, non necessariamente dello stesso tipo. Ogni referenza alle liste si può trovare al capitolo 8 del link sopracitato. Il modo più semplice per creare una lista è quello di inserire gli elementi tra parentesi quadre, esempio

$$a = [\text{“casa”}, 0, 1, [9, 1]]$$

Una cosa che ci tornerà utile è la generazione di liste di numeri interi equidistanziati. Per fare questo possiamo utilizzare la funzione **range** . La funzione range può avere un numero variabile di argomenti:

1. Un solo argomento, la funzione genera numeri da 0 all'argomento escluso. Es:

   ```
   a = range(5) = [0, 1, 2, 3, 4]
   ```

2. Due argomenti, la funzione genera numeri dal primo al secondo (escluso). Es:

   ```
   a = range(2, 5) = [2, 3, 4]
   ```

3. Tre argomenti, la funzione genera numeri dal primo al secondo (escluso) usando il terzo come passo. Es:

   ```
   a = range(2, 7, 2) = [2, 4, 6]
   ```

NumPy

NumPy è l'abbreviazione di Numerical Python e indica un modulo che permette di effettuare calcolo vettoriale e matriciale in maniera rapida. Se vogliamo utilizzare le funzioni definite in numpy dobbiamo prima importarla nel nostro notebook. La sintassi standard per fare questo è

import numpy as np.

N.B. np è l'abbreviazione standard utilizzata di solito, ma ovviamente non è importante quale nome date. Questa sintassi significa che io sto importando la libreria e nel mio codice la chiamerò np.

Come genero un array con np? Principalmente useremo tre modi:

1. np.array(object): rende array l'oggetto che passiamo, principalmente una lista. Es.

   ```
   a=np.array([1, 4, 0, 3.5, 0, 2, 6])
   ```

2. np.arange(min, max, step): genera un array di interi da min a max (escluso) a passo di step. Es.

   ```
   a=np.arange(2, 8, 2) = [2, 4, 6]
   ```

3. a=np.linspace(min, max, num): genera un array di num elementi che vanno da min a max, generalmente si usa per i float.

Per accedere all i-esimo elemento dell'array si usa la sintassi a [i] (ricordatevi che Python conta da 0!).

Gli array possono essere n dimensionali. Ad esempio se creo a=np.array([1, 2, 3], [4, 5, 6]) avrò una matrice 2x3. In questo caso, per accedere all'elemento in posizione i, j si usa l'espressione a[i, j]. Per estrarre la riga i utilizzo a[i], per estrarre la colonna i utilizzo a[:, i].

Elenchiamo adesso alcune funzioni utili. Sia a =[2, 5, 6, 1, 8, 10]:

- a.ndim: restituisce la dimensione dell'array, in questo caso 1;
- a.shape: restituisce la forma del vettore, in questo caso 1x6;
- a.size: restituisce la dimensione totale del vettore (=ndim) o della matrice;
- a.dtype: scrive il tipo numpy del dato;
- zeros(num): scrive un vettore di num elementi inizializzati a zero;
- mean(a): calcola la media di a;
- sum(a): somma gli elementi di a;
- cumsum(a): calcola la somma cumulata di a;
- std(a): calcola la deviazione standard degli elementi;
- min(a): trova il minimo;
- max(a): trova il massimo;

Le varie funzioni matematiche sono già definite in NumPy. Per sapere quali sono basta controllare la documentazione.

Una cosa parecchio interessante di NumPy è ch permette di operare con i vettori come se fossero numeri. Ad esempio, proviamo a calcolare la deviazione standard, come l'abbiamo definita a lezione, ovvero

$$\sigma = \sqrt{\sum_{i=1}^{n}(x_i - E[X])^2}.$$

Immaginiamo quindi di avere un vettore x di dimensione n che contiene diverse realizzazioni della nostra variabile aleatoria. La speranza matematica $E[X]$ non è altro che il valore atteso, nonché media, dei nostri dati. Per cui avremo

```
E[X] = np.mean(x)
```

Così facendo potremo calcolare molto rapidamente la deviazione standard usando

```
np.sqrt(np.sum((x-np.mean(x))**2))
```

Oppure, più semplicemente, usiamo

```
np.std(x)
```

Una cosa che ci tornerà molto utile poi è il modulo random di NumPy. Questo ci permette di generare numeri random. Una funzione molto utile è quella che ci permette di generare array di numeri random distribuiti uniformemente. Ci sono tre possibili modi:

1. np.random.randint(min, max, num): genera un vettore di dimensione num di numeri **interi** distribuiti uniformemente tra min e max (escluso);
2. np.random.rand(num): genera un vettore di dimensione num di numeri **reali** distribuiti uniformemente tra zero e uno (escluso);
3. np.random.uniform(min, max, num): genera un vettore di dimensione num di numeri **reali** distribuiti uniformemente tra min e max (escluso);

Funzioni che abbiamo usato o visto

True: permette di filtrare elementi dell'array che rispondono a una determinata proprietà. Dato un array a, a[condizione] sarà un array contenente solo gli elementi che verificano quella data condizione.

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
a[a > 5] = [6, 7, 8, 9, 10].
```

np.unique: prende in input un vettore e restituisce una matrice la cui prima riga contiene gli elementi del vettore e la seconda quante volte si ripetono. In realtà può fare molte più cose (vedi documentazione), per il nostro scopo basta aggiungere il flag true all'opzione return counts

```
a = [0, 3, 5, 2, 4, 4, 3, 3, 3]
F = np.unique(a, return_counts=True) = [[0, 2, 3, 4, 5],[1, 1, 4, 2, 1]]
```

np.binscount: non l'abbiamo usata ma ne abbiamo discusso a leizone. Prende un vettore e conta quante volte sono presenti i numeri interi.

np.where: è una funzione molto potente che permette di operare di operare con le condizioni. Nella sua sintassi completa, np.where cerca in un array tutti gli elementi che verificano una data condizione e gli sostituisce il valore passato come secondo argomento. Sostituisce poi il valore passato come terzo argomento agli elementi che non verificano la data condizione. Noi abbiamo usato np.where passando un unico argmento. In questo caso la funzione restituisce una matrice in cui la prima riga contiene gli indici di riga degli elementi che verificano la condizione, la seconda gli indici di colonna. Ovviamente, nel caso di un array 1D si avrà un unico indice, quello di riga.

```
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
index = np.where(a < 5) = [[0, 1, 2, 3, 4],]

b = [[0, 2, 4, 6], [1, 3, 5, 7], [2, 4, 6, 8],[3, 5, 7, 9]]
index_2 = np.where(b < 5)
        =[[0, 0, 0, 1, 1, 2, 2, 3],[0, 1, 2, 0, 1, 0, 1, 0]]
```

La seconda istruzione significa che gli elementi di b che verificano la condizione di essere minori di 5 sono: $b_{00}, b_{01}, b_{02}.b_{10}, b_{11}, b_{20}, b_{21}, b_{30}$.

SciPy.stats

Adesso concentriamoci su una libreria in cui si trovano tutte le funzioni statistiche. Va segnalato che le stesse funzioni ci possono essere anche in altre librerie, però noi ne scegliamo una che sarà quella più funzionale ai nostri scopi.

Ad esempio possiamo importare alcune distribuzioni che conosciamo, come quella binomiale, multinomiale o di Poisson

```
from scipy.stats import binom, multinomial, poisson
```

La documentazione su tutte le possibili funzioni contenute qui dentro la troviamo qui: https://docs.scipy.org/doc/scipy/reference/stats.html

Plotting

È anche importante riuscire a plottare le cose che osserviamo. Per fare questo useremo matplotlib

```
import matplotlib.pyplot as plt
```

Tutta la documentazione è qui:

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html

In generale dati due vettori x e y che contengono i valori di ascissa e i relativi valori di ordinata, la sintassi è

```
plt.plot(x, y, '.', label='')
```

dove '.' indica i punti (e permette di cambiare il tipo di punti) e label indica l'etichetta associata ai punti, qualora ne volessimo mettere una.

Per dare nome all'asse x e y utilizzeremo plt.xlabel(") e plt.ylabel("). Possiamo stabilire il range dei due assi facendo plt.xtics(start, stop, step) o plt.ytics(start, stop, step).

Spesso ci servirà disegnare degli istogrammi. Utilizzeremo la funzione hist di questa libreria, di solito in questa forma

```
plt.hist(x, bins, range, density=True)
```

dove x è il vettore di dati che vogliamo plottare come istogramma, bins può essere sia un numero intero (e in questo caso indica in quanti bin si divide l'intervallo di plotting) che una lista (e in questo caso indica gli estremi dei singoli bin), range indica il minimo e massimo range, se non è fornito prendono (x.min(), x..max()), density=true indica se l'istogramma è normalizzato o no.

In realtà possiamo fare istogrammi anche utilizzando plt.bar

```
plt.bar(x, y, width=0.5, align='center')
```

Python plotta in corrispondenza di x delle barre alte quanto il rispettivo valore presente in y. Width permette di stabilire la larghezza della barra, align l'orientamento rispetto al tick.