

Lezione 2: Numpy

Davide Evangelista

`davide.evangelista5@unibo.it`

Università di Bologna

26 Febbraio 2024

Virtual Environments

- Sconsigliato installare le librerie esterne sull'Environment principale: l'aggiornamento di una libreria potrebbe rendere necessario reinstallare Anaconda da zero.
- L'Environment principale si chiama base. Aprendo il programma Anaconda Prompt, l'Environment attivo si trova sulla sinistra della riga di comando.
- Per creare (e attivare) nuovo Environment:

```
> (base) conda create --name SN24  
> (base) conda activate SN24  
> (SN24) ...
```

Installazione librerie

- Aprire l'Anaconda Prompt (il Terminale se su Linux/MACOSX), attivare l'Environment su cui si desidera installare la libreria (e.g. numpy), e scrivere:

```
> (base) conda activate SN24
```

```
> (SN24) conda install numpy
```

- Allo stesso modo è possibile installare matplotlib, pandas e scikit-learn (sklearn).
- **Attenzione:** Non tutte le librerie sono disponibili attraverso il comando conda. Per installare librerie non presenti in conda, utilizzare pip:

```
> (SN24) pip install nome_libreria
```

- Libreria fondamentale per il calcolo matriciale e vettoriale.
- Si importa con il comando `import numpy as np`
- Tipo di dato fondamentale `numpy.array`

```
> x = np.array((1,2,3,4))
```

```
> x.size
```

```
[1] 4
```

```
> x.shape
```

```
[1] (4,)
```

```
> x.ndim
```

```
[1] 1
```

```
> x = np.array([[1,2,3],[4,5,6]])
```

```
> x.size
```

```
[1] 6
```

```
> x.shape
```

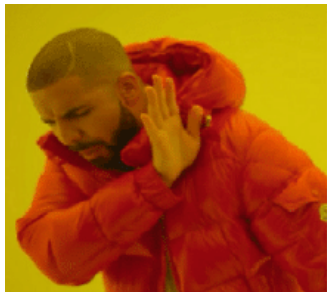
```
[1] (2, 3)
```

```
> x.ndim
```

```
[1] 2
```

Funzioni matematiche base numpy

<code>cos sin tan</code>	coseno, seno, tangente
<code>acos asin atan</code>	arcoseno, arcocoseno, arcotangente
<code>cosh sinh tanh</code>	equivalente iperbolico delle precedenti
<code>acosh asinh atanh</code>	idem
<code>log log10</code>	Logaritmo naturale, Logaritmo in base 10
<code>sqrt exp</code>	Radice quadrata, esponenziale
<code>sign abs</code>	Segno, valore assoluto
<code>round</code>	Arrotondamento all'intero più vicino
<code>floor</code>	Arrotondamento all'intero inferiore



```
1 import numpy
```



```
1 import numpy as np
```

There is no other way

Un **vettore** è una lista di oggetti numerici ordinati.

```
> x = np.array([1,3,2])
> x # crea il vettore di elementi [1 , 3 , 2]
[1] array([1, 3, 2])
> y = np.repeat(x,2)
> y # crea un vettore ripetendo due volte x
[1] array([1, 1, 3, 3, 2, 2])
> k = np.arange(1,11,1)
> k # crea un vettore che contiene i numeri da 1 a 10
[1] array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
> z = np.linspace(0,50,6) # Crea un vettore che contiene 6
elementi equispaziati tra 0 e 50
> z
[1] array([ 0., 10., 20., 30., 40., 50.] )
```

Funzioni su vettori

- `np.sort()` ordina gli elementi di un vettore in ordine crescente.
- `np.where()` restituisce le posizioni degli elementi di un vettore che rispettano una condizione.
- `np.max()`, `np.min()` restituiscono l'elemento più grande e quello più piccolo di un vettore.

```
> x = np.array([4,1,5,2,3])
> np.sort(x) # Ordina x in ordine crescente
[1] array([1, 2, 3, 4, 5])
> np.where(x >= 3) # Posizione degli elementi di x>=3
[1] (array([0, 2], dtype=int64),)
> np.max(x) # Posizione del max di x
[1] 5
> np.min(x) # Posizione del min di x
[1] 1
```


Operazioni tra vettori

- Praticamente *tutte* le operazioni standard di Python (più le funzioni base di `numpy` precedentemente descritte) si possono applicare su vettori, e vengono applicate *element-wise* (cioè elemento per elemento).

```
> x = np.array([1, 0, 1])
> y = np.array([0, -1, 1])
> x + y
[1] (array([1, -1, 2], dtype=int64), )
> x * y
[1] (array([0, 0, 1], dtype=int64), )
```

Vettori vs Vettori riga vs. Vettori colonna

Nota: I vettori hanno shape del tipo $(n,)$, che sta ad indicare che sono array unidimensionali. E' possibile rappresentare i vettori anche come:

- Vettori riga: hanno shape $(1, n)$.
- Vettori colonna: hanno shape $(n, 1)$.

Sebbene le tre rappresentazioni siano tendenzialmente equivalenti, si possono comportare diversamente in alcune situazioni specifiche:

```
> x = np.array([[1, 1, 1]]) # Ha shape (1, 3)
> y = np.array([[1], [2], [3]]) # Ha shape (3, 1)
> z = x + y # Ha shape (3, 3)
```

Vettori Booleani

- I vettori possono contenere, oltre ai numeri, dei valori booleani (True o False), alternativamente rappresentati con 0 e 1, rispettivamente.
- I vettori booleani sono generati principalmente da *operatori logici*, che vengono applicati elemento per elemento quando si confrontano due vettori (oppure un vettore e un numero).
- Due vettori booleani possono essere confrontati tra loro con *operatori di confronto*, che vengono applicati elemento per elemento, come l'operatore & (and), e | (or).

Operatori logici numpy

>, <	Maggiore, Minore
>=, <=	Maggiore o uguale, Minore o uguale
==	Uguale

Operazioni tra vettori booleani in numpy

&	and
	or
	not
^	xor

Dato un vettore x è possibile estrarne dei valori o una parte utilizzando le parentesi quadre (slicing):

- Inserendo un numero, verrà estratto il valore di x che si trova in quella posizione.
- Inserendo un vettore, verrà estratto un sottovettore di x che ha per elementi gli elementi di x con posizione corrispondente a quella descritta dal vettore.
- Inserendo un vettore booleano, verrà estratto il sottovettore di x che ha per elementi quelli in corrispondenza dei valori `True`.

Slicing (II)

```
> x = np.arange(1,100,10)
> len(x) # length restituisce il numero di elementi di x
[1] 10
> x[3]
[1] 31
> x[1:3]
[1] array([11, 21])
> x[[1, 5, 2[]]
[1]array([11, 51, 21])

# Vettore di valori Gaussiani lungo (100, )
> x = np.random.normal(0, 1, (100, ))
# Vettore Booleano con True in corrispondenza dei valori positivi
> v = x > 0
> x[v] # Ritorna solo i valori positivi di x
```

- Le matrici sono un'estensione del concetto di vettore in cui gli elementi sono disposti a forma di tabella.
- Le matrici sono array, come i vettori.
- Una matrice viene costruita con la funzione `np.array()` che prende in ingresso una lista di liste.
- La selezione degli elementi in una matrice avviene in maniera simile ai vettori. Si devono passare due valori, separati da una " , ", che indicano rispettivamente l'indice di riga e l'indice di colonna.

```
> x = np.ones(2,2) # Crea la matrice di tutti 1
> x
[1] array([[1., 1.], [1., 1.]])
> x = np.zeros(2,2) # Crea la matrice di tutti 0
> x
[1] array([[0., 0.], [0., 0.]])
> np.diag(x)
[1] array([0., 0.])
> d = np.arange(1,4)
> D = np.diag(d)
> D
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```


- `np.concatenate()` che concatena più matrici (per righe o per colonne con l'opzione `axis`) per costruire una matrice più grande.
- `A.flatten()` che presa una matrice restituisce il vettore ottenuto appiattendendo la matrice.
- `np.reshape(A, (m, n))` che ridimensiona la matrice `A` in input nella shape desiderata.

Operazioni di Algebra Lineare

Tra matrici e vettori, è possibile applicare praticamente tutte le funzioni di Algebra Lineare studiate nel rispettivo corso.

- `np.transpose()` che restituisce la trasposta della matrice in input.
- `A.T` che restituisce la trasposta della matrice `A`.
- `np.inv(A)` che calcola l'inversa di `A` (quando possibile).
- `np.dot(A, v)` che applica il prodotto riga-per-colonna tra la matrice `A` e il vettore `v`.
- `A@v` che è equivalente a `np.dot(A, v)`.

Inoltre, è bene ricordare che è possibile anche sommare (o applicare le altre operazioni base) matrici con vettori, e il risultato sarà ottenuto applicando l'operazione tra ogni riga della matrice e il vettore.

Il modulo `np.linalg`

La maggior parte delle funzioni di Algebra Lineare sono contenute nel pacchetto `np.linalg`.

- `np.linalg.rank(A)` restituisce il rango di A .
- `np.linalg.det(A)` restituisce il determinante di A .
- `np.linalg.norm(A, p)` restituisce la norma p di A .
- `np.linalg.norm(A, 'fro')` restituisce la norma Frobenius di A .

Slice di array

- Per estrarre la riga i -esima dall'array A si usa il comando $A[i, :]$
- Per estrarre più righe consecutive dall'array A si usa il comando $A[i:j, :]$
- Per estrarre le colonne la cosa è analoga cambiando gli indici, ossia $A[:, i:j]$
- Per estrarre un singolo elemento in posizione (i, j) dall'array A si usa il comando $A[i, j]$
- Per estrarre una sottomatrice si usa il comando $A[i1:i2, j1:j2]$

Il modulo `np.random`

La maggior parte delle funzioni `numpy` dedicate alla probabilità e la statistica si trovano all'interno del modulo `np.random`:

- `np.random.normal(mean, std, size)` costruisce un'array di valori Gaussiani con media e deviazione standard fissate, con la `size` data in input.
- `np.random.uniform(a, b, size)` costruisce un'array di valori Uniformi in $[a, b]$ con media e deviazione standard fissate, con la `size` data in input.
- `np.mean()`, `np.std()`, `np.var()` restituisce media, deviazione standard e varianza dell'array preso in input.

- Lavorando con dati di grandi dimensioni, è difficile ricordarsi la posizione esatta dei valori all'interno di una matrice e/o un vettore.
- Farebbe comodo assegnare un nome ad ogni elemento, che si potrà poi utilizzare per richiamarlo.
- Ci viene in aiuto la libreria Pandas.
- Si importa con il comando `import pandas as pd`

```
> import numpy as np
> import pandas as pd
> d = np.arange(1,4)
> D = np.diag(d)
> df = pd.DataFrame(D, columns = ['C_A', 'C_B', 'C_C'])
```

Dataframe

- I Dataframe sono il tipo di dato ideale quando si lavora con i dati.
- Un Dataframe è una tabella di valori misti con n righe e p colonne.
- Il valore n rappresenta il numero di osservazioni di un fenomeno, mentre p il numero di parametri di ciascuna osservazione.

```
> Data = pd.DataFrame({'nome': ['Mario', 'Luca'],  
  'cognome': ['Rossi', 'Bianchi'],  
  'età': [45, 38]})
```

```
> Data
```

```
nome cognome eta
```

```
0 Mario    Rossi  45
```

```
1 Luca  Bianchi  38
```

Dataframe: Estrazione celle

- Per estrarre una colonna `Data['nomecolonna']`.
- Stessa cosa con `Data.loc[:, 'nomecolonna']`.
- È possibile estrarre e modificare il valore di una qualsiasi cella.

```
> Data['nome']  
[1] "Mario" "Luca"  
> Data['età']  
[1] 45 38  
> Data['età'][1]  
[1] 38  
> Data.loc[1,'età']=12  
> Data  
nome cognome eta  
1 Mario    Rossi  45  
2  Luca  Bianchi  12
```


Dataframe: Subframe

- È possibile estrarre un sottinsieme del dataframe che verifichi certe condizioni.

```
> Data
```

```
nome cognome eta
```

```
1  Marco    Rossi  15
```

```
2  Mario  Bianchi  45
```

```
3   Luca    Verdi  38
```

```
4  Piero    Rossi  60
```

```
5 Davide  Bianchi  80
```

```
> subData = Data[Data['cognome']=='Rossi']
```

```
> subData[['nome', 'cognome']]
```

```
nome cognome
```

```
1 Marco    Rossi
```

```
4 Piero    Rossi
```

Dataframe: Eliminare colonne

- È possibile eliminare una colonna del dataframe con il metodo drop.

```
> Data
```

```
nome cognome eta
```

```
1  Marco    Rossi   15
```

```
2  Mario  Bianchi   45
```

```
3   Luca    Verdi   38
```

```
4  Piero    Rossi   60
```

```
> Data = Data.drop(columns=['età'])
```

```
> Data
```

```
nome cognome
```

```
1  Marco    Rossi
```

```
2  Mario  Bianchi
```

```
3   Luca    Verdi
```

```
4  Piero    Rossi
```

Dataframe: valori NaN

- Dataset reali possono contenere valori NaN.
- È possibile rimuovere le osservazioni (righe) del dataframe che contengono valori NaN con il metodo `dropna`.

```
> Data
```

```
nome cognome eta
```

```
1  Marco    Rossi  15
2  Mario  Bianchi NaN
3  NaN     Verdi  38
4  Piero    Rossi  60
```

```
> Data.dropna()
```

```
nome cognome eta
```

```
1  Marco    Rossi  15
4  Piero    Rossi  60
```

Dataframe: CSV

- Importeremo dataset preesistenti salvati in file csv o txt.
- Per leggere un file csv si utilizza il comando `read_csv()`.
- Per salvare un file csv si utilizza il comando `to_csv()`.
- Tutto viene scritto/letto dalla working directory corrente.

```
> Data
```

```
nome cognome eta
```

```
1  Marco    Rossi   15
```

```
2  Piero    Rossi   60
```

```
> Data.to_csv('data.csv')
```

```
> del Data
```

```
> Data
```

```
NameError: name 'Data' is not defined
```

```
> pd.read_csv('data.csv')
```

```
X   nome cognome eta
```

```
1 1  Marco    Rossi   15
```

```
2 2  Piero    Rossi   60
```

Variabili quantitative vs categoriche

In Statistica, si distinguono due principali gruppi di variabili:

- **Numeriche o Quantitative:** tutte quelle variabili esprimibili direttamente come numeri. Un esempio di variabili numeriche sono l'età, l'anno corrente, il numero di chilometri percorso o il punteggio assegnato ad una recensione.
- **Categoriche o Qualitative:** tutte quelle variabili che non sono esprimibili da numeri, ma solo descritte da stringhe. Un esempio sono il nome, il cognome, il sesso, la cittadinanza o il nome di un piatto.

Per gestire variabili di tipo categorico in Pandas si usa il tipo `category` che associa ad ogni categoria un valore/chiave.

Variabili categoriche

```
> df = pd.DataFrame({"A": ["a", "b", "c", "a"]})
> df["A"] = df["A"].astype("category")
> df["A"]
[1] 0      a
1     b
2     c
3     a
Name: A, dtype: category
Categories (3, object): ['a', 'b', 'c']
```

- Il metodo `head(n)` restituisce le prime `n` righe del DataFrame.
- Il metodo `tail(n)` restituisce le ultime `n` righe del DataFrame.
- Il metodo `describe()` restituisce le statistiche descrittive del DataFrame.