Laboratorio Apprendimento Automatico

Laurea Magistrale in Informatica

Mirko Polato
14 novembre 2016

Python

Perché python?

- Molti moduli per l'analisi e la visualizzazione dei dati pre esistenti;
- Facile da imparare;
- Efficiente;
- Universale.

L'ecosistema scientifico di python

- **Numpy**: computazione numerica (i.e., matrici multidimensionali);
- **Scipy**: funzioni numeriche di alto livello (i.e., regressione, ottimizzazione), strutture *sparse*;
- **Matplotlib**: visualizzazione 2D (e 3D) dei dati.

Numpy

Importare il modulo numpy

```
>>> import numpy as np
```

l'aggiunta dell'alias **np** è ormai una convenzione universalmente accettata.

Array e inizializzazione

Creazione array da lista di valori:

```
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
```

Creazione array da **range**:

```
>>> a = np.array(range(4))
>>> a
array([0, 1, 2, 3])
```

Creazione array specificandone il tipo di valori:

```
>>> a = np.array(range(4), dtype=float)
>>> a
array([0., 1., 2., 3.])
```

Matrici

La creazione di una matrice è simile a quella di un array (che è in effetti una matrice colonna) dove la lista di valori è una lista di liste:

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

Inizializzazione matrici

Esistono molte matrici 'standard' offerte da numpy:

```
>>> z = np.zeros((3,4)) #matrice 3x4 con tutti zeri
>>> o = np.ones((3,4))  #matrice 3x4 con tutti uno
>>> i = np.identity(4)  #matrice 4x4 con tutti zeri e 1
    sulla diagonale
>>> e = np.eye(4, 1)    #come identita' ma con diagonale
     'spostata'
```

Array info

Esistono diverse funzioni utili per ottenere informazioni sugli array:

```
>>> mat = np.array([[1,2,3], [4,5,6]], dtype=float)
>>>
>>> mat.dtype #ritorna il tipo dei valori
dtype('float64')
>>> mat.ndim #ritorna il numero di dimensioni
2
>>> mat.shape #ritorna la forma della matrice
(2,3)
>>> len(mat) #ritorna il numero di righe
2
>>> mat.size #ritorna il numero di elementi
6
```

Modifica degli array

Gli array si possono indicizzare (e modificare) come le liste:

```
>>> a = np.array(np.arange(0, 1, 0.25))
>>> a[2]
0.5
>>> a[1:]
array([ 0.25,  0.5 ,  0.75])
>>> a[2] = 6
>>> a
array([ 0.  ,  0.25,  6.  ,  0.75])
```

E' inoltre possibile modificarne la forma:

```
>>> m = np.array([[0,1,2,3,5],[6,7,8,9,10]])
>>> m.shape
(2,5)
>>> m1 = m.reshape((5,2)) #ritorna un nuovo array!!!
>>> m1.shape
(5,2)
```

Altri metodi utili

- **sum()**: ritorna la somma degli valori;
- **prod()**: ritorna il prodotto degli valori;
- **min()**: ritorna il minimo valore;
- **max()**: ritorna il massimo valore;
- **mean()**: ritorna la media dei valori.

Vedere anche **var()**, **std()**, **argmin()** e **argmax()**.

Operazione tra array

Operazioni classiche:

```
>>> a = np.array(range(10)) #vettore riga
>>> b = np.ones((1,10)) #vettore riga
>>> a+b  #simile a-b
array([[1., 2., 3., 4., 5., 6., 7., 8., 9., 10.]])
>>> a*b  #simile a/b
array([[0., 1., 2., 3., 4., 5., 6., 7., 8., 9.]])
```

Una delle operazioni non standard tra array (compatibili) più usata è il prodotto scalare:

```
>>> np.dot(a, b.transpose()) #dot product
array([ 45.])
```

Notare che **b.transpose()** è necessario perché il dot product si esegue tra vettore riga e vettore colonna.

Algebra lineare

Numpy contiene moduli ultimi per l'algebra lineare, come ad esempio **linalg**.

```
>>> q = np.array([[4,2,0], [9,3,7], [1,2,1]], float)
>>> np.linalg.det(q) #calcola il determinante
-48.0
>>> evals, evec = np.linalg.eig(q) #autovalori e
    autovettori
>>>
>>> qinv = np.linalg.inv(q) #calcola l'inversa
>>> np.dot(q, qinv) #test
array([[ 1.000000e+00,  0.000000e+00, -2.220446e-16],
       [ 0.000000e+00,  1.000000e+00,  0.000000e+00],
       [ 0.000000e+00,  0.000000e+00,  1.000000e+00]])
```

Random

Numpy offre inoltre il modulo random per la generazioni di numeri casuali:

```
>>> np.random.random() #numero casuale tra [0,1]
>>> 0.530586755605294 1
>>> np.random.seed(17)
>>> np.random.random() #numero 'casuale' fissato dal
    seed
0.2946650026871097
```

Estrazioni di campioni da una popolazione:

```
>>> random.sample(range(1000), 100)
```

Si possono generare numeri casuali secondo distribuzioni fissate:

```
>>> np.random.normal(0, 1, size=10)
```

genera una sequenza di 10 numeri casuali secondo la distribuzione normale con $\mu = 0$ e $\sigma^2 = 1$.

Scipy

- **scipy.constants**: costanti matematiche e fisiche;
- **scipy.special**: funzioni in uso in fisica matematica (e.g., coefficiente binomiale);
- **scipy.integrate**: metodi di integrazione numerica (e.g., trapezoidale, Simpson), integrazione di equazioni differenziali;
- **scipy.optimize**: metodi di ottimizzazione (minimi quadrati, gradiente, simulated annealing);
- **scipy.linalg**: estensione di *numpy.linalg* - soluzione di sistemi lineari, calcolo matriciale, decomposizione, fattorizzazione;
- **scipy.sparse**: gestione di matrici sparse.

Matplotlib

E' una libreria orientata agli oggetti per la creazione di grafici. In particolare noi useremo il modulo **pyplot** che fornisce un framework con interfaccia MATLAB-like.

Importare il modulo:

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

ci riferiremo a **pyplot** con **plt**.

Creare un grafico

Creare un semplice grafico:

```
>>> x = np.array(range(50))
>>> y = x**2
>>> plt.plot(x,y)
```

Il grafico è stato creato ora non resta che visualizzarlo:

```
>>> plt.show()
```

Grafico con più curve

E' sufficiente aggiungere altri **plot** prima della visualizzazione:

```
>>> x = np.array(range(50))
>>> y = x**2
>>> y1 = x**1.8
>>> plt.plot(x,y)
>>> plt.plot(x,y1)
>>> plt.show()
```

I colori e gli stili vengono automaticamente scelti.

Abbellimento - 1

Iniziamo con il definire cosa vogliamo visualizzare:

```
>>> x = np.linspace(-np.pi, np.pi, 256)
>>> c, s = np.cos(x), np.sin(x)
```

`x` è un array di 256 valori equidistanti nell'intervallo $[-\pi, \pi)$. `c` ed `s` saranno i corrispettivi valori di *coseno* e *seno*. Proviamo a plottare:

```
>>> plt.plot(x, c)
>>> plt.plot(x, s)
>>> plt.show()
```

Bello ma non bellissimo.

Abbellimento - 2

Proviamo a cambiare colore e stile delle curve:

```
>>> plt.plot(x, c, color="blue", linestyle="-")
>>> plt.plot(x, s, color="red", linestyle="--")
>>> plt.show()
```

Aggiungiamo la legenda in alto a sinistra:

```
>>> plt.plot(x, c, color="blue", linestyle="-", label="
    Seno")
>>> plt.plot(x, s, color="red", linestyle="--", label="
    Coseno")
>>> plt.legend(loc='upper left')
>>> plt.show()
```

Molto meglio!!

Abbellimento - 3

Altre funzioni utili per modificare i dettagli del grafico:

- **xlim, ylim**: modifica i limiti di visualizzazione per gli assi x e y;
- **xticks, yticks**: definisce i valori da visualizzare negli assi;
- **xlabel, ylabel**: aggiunge un'etichetta agli assi;
- **xscale, yscale**: modifica la scala degli assi;
- **title**: aggiunge un titolo al grafico.

Miscellanea

Altre funzionalità degne di nota:

- **subplot**: permette di mostrare più grafici all'interno di una stessa immagine;
- **matshow**: metodo che visualizza il contenuto di una matrice secondo una codifica a colori;
- **hist**: visualizza un grafico a istogrammi.