numpy

January 28, 2024

0.1 Uvod v `numpy`: tabele, indeksiranje, operacije

Bistvo tegale bo spoznavanje modula `numpy` prek nalog iz Advent of Code 2021. Letos je bilo precej nalog, ki se jih je elegantno rešilo s tem modulom. Izbral sem jih in jih uredil glede na to, kaj iz `numpy`-ja potrebujemo. Pred vsako nalogo bom razložil nekaj stvari, ki nam pridejo prav. Potem je ideja, da rešitev nalogo. Potem pa pogledate mojo rešitev. Po tem je vedno koristno, če še sami (recimo po spominu, brez škiljenja) rešite nalogo z enakim trikom, če ste jo prej kako drugače (sploh, če je bil tisti "kako drugače" nepotrebno zapleten).

Modula numpy navadno ne uvozimo z `import numpy`, temveč z

[1]:
```
import numpy as np
```

Tako počnejo praktično vsi. To pa zato, ker njegove funkcije kličemo res velikokrat in se zelo pozna, ali moramo pisati, na primer, `numpy.sum` ali `np.sum`. Sploh, če imamo `numpy.sum(numpy.any(~numpy.any(x, axis=1)))`.

`numpy` je modul za delo s tabelami. (Včasih se mi bo zareklo in bom namesto "tabela" rekel matrika. To ni najbolj lepo, ni pa vedno narobe.) Te so lahko poljubno velike, poljubno dimenzionalne in vsebujejo poljubne reči. Za razliko od Pythonovih seznamov `numpy`-jeve tabele vedno vsebujejo elemente istega tipa - same `int`-e, same `float`-e, same `bool`-e…

Tabelo najprejprosteje pripravimo s funkcijo `numpy.array`, ki ji kot argument podamo običajni Pythonov seznam. Če vsebuje same `int`-e, bo to tabela `int`-ov, če bo vmes kakšen `float`, bo tabela `float`-ov, če bodo sami `bool`-i, bo tabela `bool`-ov. Da se še kaj drugega, vendar nas za zdaj ne zanima.

[56]:
```
a = np.array([5, 8, 3, 1, 10, 5])
```

[57]:
```
a
```

[57]:
```
array([ 5,  8,  3,  1, 10,  5])
```

Tale `a` se obnaša nekoliko podobno kot Pythonov seznam. Lahko ga indeksiramo - z leve, z desne in z vsakršnimi rezinami.

[58]:
```
a[2]
```

[58]:
```
3
```

[59]:
```
a[-3]
```

[59]: 1

[60]:
```
a[2:5]
```

[60]:
```
array([ 3,  1, 10])
```

Tako kot v Pythonovih seznamih lahko vse elemente razen prvega dobimo z

[68]:
```
a[1:]
```

[68]:
```
array([ 8,  3,  1, 10,  5])
```

in vse razen zadnjega z

[69]:
```
a[:-1]
```

[69]:
```
array([ 5,  8,  3,  1, 10])
```

Od Pythonovih seznamov pa se razlikuje po tem, da računske operacije nad njim delujejo "po elementih".

[62]:
```
a
```

[62]:
```
array([ 5,  8,  3,  1, 10,  5])
```

[63]:
```
a + 10
```

[63]:
```
array([15, 18, 13, 11, 20, 15])
```

[64]:
```
a * 7
```

[64]:
```
array([35, 56, 21,  7, 70, 35])
```

[65]:
```
a ** 2
```

[65]:
```
array([ 25,  64,   9,   1, 100,  25])
```

In celo

[25]:
```
2 ** a
```

[25]:
```
array([  32,  256,    8,    2, 1024,   32])
```

Še bolj pomembno: tudi računske operacije med dvema tabelama se odvijajo po elementih.

[44]:
```
b = np.array([4, 1, 12, 2, 1, -5])
```

[45]:
```
a
```

[45]: array([5, 8, 3, 1, 10, 5])

[46]:
```
b
```

[46]: array([4, 1, 12, 2, 1, -5])

[47]:
```
a - b
```

[47]: array([1, 7, -9, -1, 9, 10])

[48]:
```
a * b
```

[48]: array([20,   8, 36,   2, 10, -25])

Za tole je pomembno, da sta tabeli enako veliki. Če ne bi bili, bi se numpy pač pritožil.

Podobno se vedejo operacije, kot so `<` in `==`.

[49]:
```
b == 1
```

[49]: array([False, True, False, False, True, False])

[50]:
```
a > b
```

[50]: array([True, True, False, False, True, True])

Kako ugotoviti, koliko `1` je v tabeli `b`? (Dve sta.) Tabele nimajo metode `count`. Je ne potrebujejo. Naredimo tako:

[54]:
```
sum(b == 1)
```

[54]: 2

Vendar tega ne počnemo - nikoli. `numpy` ima svoj `sum`, `np.sum`, ki je veliko hitrejši od Pythonovega. Pa še nekatere superpowers ima. Poklicati moramo torej

[55]:
```
np.sum(b == 1)
```

[55]: 2

Rezultat je isti in tudi pri hitrosti se zdaj seveda še ne pozna. Nekoč pa se bo, zato se naučite delati tako.

Še eno vprašanje, preden se lotimo naloge: kako bi ugotovili, koliko elementov `a` je večjih od istoležnih elementov `b`?

[66]:
```
a > b
```

[66]: array([True, True, False, False, True, True])

[67]:
```
np.sum(a > b)
```

[67]: 4

0.2 Naloga

Zdaj znate dovolj za elegantno reševanje prve naloge, Sonar Sweep.

Lahko jo rešitev najprej v čistem Pythonu, vendar jo rešite lepo - do zaporednih elementov pridite z `zip`-om in rezinami in tako naprej. Potem razmislite, kako se to pove v `numpy`-ju. Če izvzamemo pripravo podatkov, se pravi, ko so podatki enkrat že v tabeli, bi moral biti prvi del zlahko rešljiv z enim samim klicem … drugi del pa tudi, če se znajdete. Če ne, bo pa kakšna vrstica več.