numpy-1-del

January 28, 2024

0.1 Range

Ko sem prejle rekel, da tabelo najpreprosteje naredimo z `np.array`, sem vedel, da to ni čisto res. Preprostejši (a še vedno ne najpreprostejši) način je funkcija `np.arange`. Ta deluje tako kot Pythonov `range`, le da vrne numpyjevo tabelo.

[1]:
```
import numpy as np

np.arange(5)
```

[1]:
```
array([0, 1, 2, 3, 4])
```

Morda vam pride prav pri naslednji nalogi. Morda pa ne, kakor se boste lotili.

0.2 Večdimenzionalne tabele

Rekel sem, da so Pythonove tabele lahko tudi večdimenzionalne. V Pythonu lahko naredimo seznam seznamov. Če ga podamo numpyjevemu `np.array`, dobimo dvodimenzionalno tabelo.

[2]:
```
a = np.array([[5, 8, 7, 1, 3],
              [2, 5, 1, 1, 4],
              [7, 6, 1, 9, 2]])
```

Vsi seznami znotraj seznama morajo biti seveda enako dolgi. Numpyjev array bo vedno pravokoten, ne nazobčan.

Pri takem seznamu nas bo pogosto zanimala njegova oblika - koliko vrstic in koliko stolpcev ima. To lahko izvemo, tako kot če bi šlo za seznam seznamov, z `len`.

[3]:
```
len(a)
```

[3]:
```
3
```

[4]:
```
len(a[0])
```

[4]:
```
5
```

Vendar to potrebujemo pogosto in posebej pri večdimenzionalnih tabelah bi postalo hitro zoprno. Namesto `len` uporabimo tabelin atribut `shape`:

[5]:
```
a.shape
```

[5]:
```
(3, 5)
```

`a` ima torej tri vrstice in pet stolpcev.

Pazite, `shape` ni funkcija. Tega `shape` ne pokličemo, temveč že vsebuje vrednost.

0.2.1 Indeksiranje večdimenzionalnih tabel

Dvodimenzionalno tabelo indeksiramo z dvema indeksoma.

[6]:
```
a[2, 3]
```

[6]:
```
9
```

Če navedemo le en indeks, dobimo vrstico.

[7]:
```
a[2]
```

[7]:
```
array([7, 6, 1, 9, 2])
```

To je isto, kot če bi napisali

[8]:
```
a[2, :]
```

[8]:
```
array([7, 6, 1, 9, 2])
```

Se pravi, želimo drugo vrstico - in to celo.

Lahko pa tudi obrnemo in zahtevamo drugi stolpec.

[9]:
```
a[:, 2]
```

[9]:
```
array([7, 1, 1])
```

Ali pa vse stolpce od drugega naprej.

[10]:
```
a[:, 2:]
```

[10]:
```
array([[7, 1, 3],
       [1, 1, 4],
       [1, 9, 2]])
```

Vse vrstice do druge, vendar le zadnje tri stolpce.

[11]:
```
a[:2, -3:]
```

[11]:
```
array([[7, 1, 3],
       [1, 1, 4]])
```

Skratka, počnemo lahko vse, kar smo počeli, ko smo indeksirali v eni dimenziji, le da to počnemo v dveh, z dvema indeksoma, ki sta lahko števili ali pa rezine.

0.2.2 Operacije po oseh

Kakšna je vsota `a`?

[12]:
```
np.sum(a)
```

[12]:
```
62
```

To je sicer res, vendar to pogosto ni to, kar bi hoteli. Kaj če bi hoteli vsote vseh stolpcev ali vrstic?

Tabela je dvodimenzionalna. Dimenzije štejemo od 0, tako kot se pač vedno šteje. Navpična dimenzija (vrstice) je 0, vodoravna (stolpci) je 1. Funkciji `sum` lahko dodamo še dimenzijo oziroma os, *axis*, po kateri naj sešteva. Seštevamo lahko torej navpično in dobimo vsoto stolpcev:

[13]:
```
np.sum(a, axis=0)
```

[13]:
```
array([14, 19,  9, 11,  9])
```

ali vodoravno, da dobimo vsote vrstic:

[14]:
```
np.sum(a, axis=1)
```

[14]:
```
array([24, 13, 25])
```

0.3 Naloga

Naloga, ki jo bomo reševali s tem znanjem, je **prvi del** tretje naloge Binary Diagnostic. Trik je v tem, da poskusite poiskati najpogostejši bit za vse stolpce hkrati. Če ne znate, pa najprej vsak stolpec posebej, pa bomo skupaj pogledali, kako to narediti po večih.

Še dva nasveta. Če boste kdaj imeli tabelo `bool`-ov, na primer

[15]:
```
m = np.array([False,  True, False, False,  True, False])
```

in se vam jo bo zahotelo spremeniti v tabelo `int`-ov, pokličite

[16]:
```
m.astype(int)
```

[16]:
```
array([0, 1, 0, 0, 1, 0])
```

`astype` seveda ne spreminja tabele, temveč vrne novo tabelo.

Drugi nasvet pa povezuje nekaj iz te naloge in nekaj, o čemer smo pisali pri prejšnji. Tole vam zna priti prav:

[17]:
```
2 ** np.arange(10)
```

[17]:
```
array([  1,   2,   4,   8,  16,  32,  64, 128, 256, 512])
```