

Dan 3: Binary Diagnostic, 1. del

V nalogi dobimo seznam števil v dvojiškem zapisu. Odkriti moramo, kakšna je najpogostejša vrednost prvega bita, drugega bita in tako naprej do konca. Če je obojih enako, rečemo, da je pogostejša enica. Te, najpogostejše vrednosti sestavimo v število. Se pravi, vsak bit novega števila izračunamo kot najpogostejšo vrednost bit na tem mestu med vsemi števili na seznamu.

To število moramo nato pomnožiti s komplementom tega števila.

Podatki so videti takole:

```
00100
11110
10110
10111
10101
01111
00111
11100
10000
11001
00010
01010
```

Preberimo jih v tabelo: odpremo datoteko, gremo z zanko `for` čez vrstice, vsaki vrstici odluščimo `\n`, nato pa gremo čez znake te vrstice in vsakega posebej spremenimo v `int`. Vse skupaj zložimo v seznam in ga damo funkciji `np.array`.

```
import numpy as np
```

```
data = np.array([[int(x) for x in v.strip()] for v in open("example.txt")])
data
```

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

Mimogrede pogledajmo še dimenzije naše tabele:

```
data.shape
```

```
(12, 5)
```

Prva dimenzija ima velikost 12; imamo 12 vrstic. Druga je 5, 5 stolpcev oziroma, v naši zgodbi, bitov.

Najpogostejšo vrednost v vrstici lahko dobimo, recimo, tako da izračunamo vsoto po osi (dimenziji) 0, torej po vrsticah.

```
np.sum(data, axis=0)
array([7, 5, 8, 7, 5])
```

Enica je večinska vrednost, kadar je ta vsota večja (ali enaka) od polovice števila vrstic. To dobimo z `len(data)` ali z `data.shape[0]`. `data.shape` je namreč terka, ki vsebuje dimenziji tabele.

```
np.sum(data, axis=0) >= data.shape[0] / 2
array([ True, False,  True,  True, False])
```

Kar smo pravkar naračunali, poimenujmo `bits`. To sicer ni potrebno, a da bomo lepše videli, kaj se dogaja, pretvorimo tabelo iz `bool` v `int`.

```
bits = (np.sum(data, axis=0) >= data.shape[0] / 2).astype(int)
```

```
bits
array([1, 0, 1, 1, 0])
```

Namesto seštevanja in primerjanja z dolžino, bi lahko s funkcijo `np.mean` izračunali poprečno vrednost in preverjali, ali je večje ali enako 0.5.

Tudi to, kar sledi - pretvarjanje v desetiški zapis - bi lahko naredili na več načinov. Tu bomo uporabili najbolj zabavnega in poučnega. Glede na gornjo tabelo je število, ki ga iščemo, enako

```
1 * 16 + 0 * 8 + 1 * 4 + 1 * 2 + 0 * 1
```

Zmnožiti moramo torej naslednji tabeli

```
[ 1, 0, 1, 1, 0]
[16, 8, 4, 2, 1]
```

Prvo imamo, drugo moramo pripraviti. Kako pripravimo tabelo potenc dvojke?

Najprej pripravimo števila od 0 do toliko, kolikor bitov imamo. Število bitov, kot smo videli, izvemo v `data.shape[1]`

```
np.arange(data.shape[1])
array([0, 1, 2, 3, 4])
```

Izračunati moramo tabelo `[2 ** 0, 2 ** 1, 2 ** 2, ...]`. Se pravi

```
2 ** np.arange(data.shape[1])
array([ 1,  2,  4,  8, 16])
```

Vse skupaj moramo še obrniti, saj mora biti najpomembnejši bit, 16, na levi. Potence shranimo v `powers`.

```
powers = 2 ** np.arange(data.shape[1])[:-1]
```

```
powers
```

```
array([16,  8,  4,  2,  1])
```

Če tabeli zmnožimo, dobimo

```
powers * bits
```

```
array([16,  0,  4,  2,  0])
```

To moramo le še sešteti.

```
np.sum(powers * bits)
```

```
22
```

Komplement bomo dobili z `1 - bits` (če tabele ne bi spreminjali iz `bool` v `int`, pa bi lahko uporabili tudi operator za dvojiški komplement, `~bits`).

```
np.sum(powers * (1 - bits))
```

```
9
```

Celotna rešitev prvega dela je torej

```
bits = (np.sum(data, axis=0) >= data.shape[0] / 2).astype(int)
```

```
np.sum(powers * bits) * np.sum(powers * (1 - bits))
```

```
198
```