lektion12

January 15, 2025

1 Schreiben und Lesen von Daten

Erstellen eines Beispieldatensatzes

[4]:
```
data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
data
```

[4]:
```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

1.1 Schreiben und Lesen von Daten mit Python

Hier verwenden wir die in Python eingebaute Funktion **open**, um auf Dateiobjekt (Fileobject) zu erzeugen.

```
open(file, mode)
mode : 'w' schreiben (write)
       'a' anhängen  (append)
       'r' lesen     (read)
file : Name der Datei
```

[1]:
```
f = open('daten.txt', 'w')
f.close()
```

[2]:
```
f.
```

[2]:
```
True
```

1.1.1 Speichern von Daten in einer Textdatei

Die ganzzahligen Einträge werden mit der Methode **write** als String durch Kommas getrennt und zeilenweise in die Datei data.txt eingetragen

[5]:
```
with open('data1.txt', 'w') as file:
    for zeile in data:
        for elem in zeile[:-1]:
            file.write(f'{elem},')
        file.write(f'{zeile[-1]} \n')  # '\n' sorgt fuer eine neue Zeile
```

Mit dem **with** Statement lassen sich die folgenden Befehle gut zusammenfassen. Der Kontextmanager sorgt in dem Fall dafür, dass man das Fileobjekt nicht abschliessen muss und am Ende "aufgeräumt" wird.

Mit drei Zeilen mehr kann man auch selbst dafür sorgen, dass bei einem Fehler aufgeräumt wird.

[65]:
```
file = open('data1a.txt', 'w')
try:
    for zeile in data:
        for elem in zeile[:-1]:
            file.write(f'{elem},')
        file.write(f'{zeile[-1]} \n')  # '\n' sorgt fuer eine neue Zeile
finally:
    file.close()
```

1.1.2 Lesen von Daten aus einer Textdatei

Die Datei 'data.txt' wird zeilenweise eingelesen. Es wird angenommen, dass in jeder Zeile die ganzzahligen Elemente durch Kommas getrennt sind.

[57]:
```
loaded_data_txt = []
with open('data1.txt', 'r') as file:
    for line in file:
        l = line.strip().split(',')
        loaded_data_txt.append([int(elem) for elem in l])

loaded_data_txt
```

[57]:
```
[[1, 2, 3], [4, 5, 6], [7, 8, 0]]
```

1.2 Schreiben und Lesen von Daten mit NumPy

Für matrixartige Daten geht das in NumPy einfacher

[15]:
```
import numpy as np
```

[35]:
```
data
```

[35]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

1.2.1 Speichern eines NumpyArrays in einer Textdatei

Hierbei sollen als Trennzeichen wieder Kommas verwendet werden

[36]:
```
np.savetxt('data2.txt', data, delimiter=',')
```

numpy wandelt hier die Daten ungefragt in float um

[37]:
```
np.savetxt('data3.txt', data, fmt='%d', delimiter=',')
```

die Angabe des Formats ‘d’ sorgt dafür dass keine unnötigen Kommazahlen erzeugt werden

1.2.2 Lesen von Daten aus einer Textdatei als NumPyarray

[38]:
```
loaded_data2_txt = np.loadtxt('data2.txt', delimiter=',')
loaded_data2_txt
```

[38]:
```
array([[1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.]])
```

[39]:
```
loaded_data3_txt = np.loadtxt('data3.txt', dtype='int', delimiter=',')
loaded_data3_txt
```

[39]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Speichern und Lesen von Daten in einer Binärdatei (NumPy-Format)

[40]:
```
np.save('data.npy', data)
```

[41]:
```
loaded_data_npy = np.load('data.npy')
loaded_data_npy
```

[41]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

1.3 Speichern und Lesen von mehreren Arrays in einer einzigen Binärdatei

[42]:
```
extra_data = np.array([10, 11, 12])
np.savez('data_multiple.npz', array1=data, array2=extra_data)
```

[43]:
```
loaded_data_npz = np.load('data_multiple.npz')
print("Array 1:")
print(loaded_data_npz['array1'])
print("Array 2:")
print(loaded_data_npz['array2'])
```

```
Geladene Arrays aus 'data_multiple.npz':
Array 1:
[[1 2 3]
 [4 5 6]
```

```
 [7 8 9]]
Array 2:
[10 11 12]
```

[ ]: