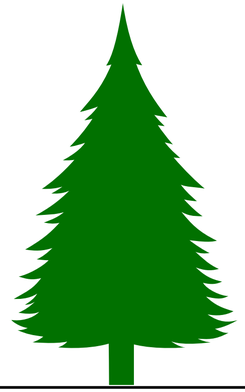
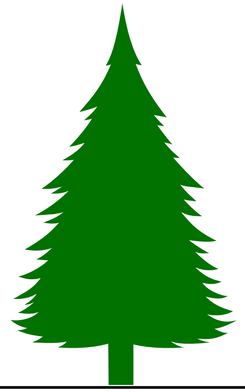
Werkzeuge für das wissenschaftliche Arbeiten

Python for Machine Learning and Data Science

Magnus Bender
bender@ifis.uni-luebeck.de
Wintersemester 2023/24





Inhaltsübersicht

1. Programmiersprache Python
   a) *Einführung, Erste Schritte*
   b) *Grundlagen*
   c) *Fortgeschritten*
2. Auszeichnungssprachen
   a) *LaTeX, Markdown*
3. Benutzeroberflächen und Entwicklungsumgebungen
   a) *Jupyter Notebooks lokal und in der Cloud (Google Colab)*
4. Versionsverwaltung
   a) *Git, GitHub*
5. Wissenschaftliches Rechnen
   a) **NumPy, SciPy**
6. Datenverarbeitung und -visualisierung
   a) Pandas, matplotlib, NLTK
7. Machine Learning (scikit-learn)
   a) Grundlegende Ansätze (Datensätze, Auswertung)
   b) Einfache Verfahren (Clustering, …)
8. DeepLearning
   a) TensorFlow, PyTorch, HuggingFace Transformers

Themen

I. Projektaufgabe 3

    1. Herangehensweise & Tipps

II. Wissenschaftliches Rechnen

    1. NumPy

    2. SciPy

III. Reguläre Ausdrücke

Projektaufgabe 3

„Git, Markdown und LaTeX“

1. Git-Repository clonen
2. Readme-Datei
3. LaTeX
    - *Gutes* LaTeX
    - Grafik
    - Temporäre Dateien
4. Python-Code
    - Versionen aufräumen
    - Branch mergen
5. Push auf GitHub
6. Abgabe (im Moodle)

Herangehensweise & Tipps

- Schrittweise vorgehen
- LaTeX
  - Beispiele aus Vorlesung nutzen
  - Inhalt soll bleiben, Form kann und soll leicht abweichen
  - Temporäre Dateien löschen
- Regelmäßig Commits machen
- Eigenes Repository als weiteres Remote hinzufügen
- Den richtigen Branch in den richtigen Branch mergen

II.
Wissenschaftliches Rechnen

1. NumPy

Installation

Wir verlassen nun die Python-Standardbibliothek

- Python Paket
  - https://numpy.org/
- Installation z.B. mit `pip3 install numpy`
- Import

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

Üblich: Import mit kurzem Namen

Warum NumPy?

- Matrix- und Array-Operationen (N-dimensional)
- Mathematische und Numerische (Standard-)Funktionen
- Hohe Performance

Intern in C-Code

- Verbindung und Nutzung in anderen Paketen (z.B. SciPy)

Arrays

```
import numpy as np

one = np.array([1, 2, 3, 4, 5, 6])
two = np.array([
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9, 10, 11, 12]])

print(one)        [1 2 3 4 5 6]

print(two)        [[ 1  2  3  4]
                   [ 5  6  7  8]
                   [ 9 10 11 12]]

print(one.shape)      (6,)

print(two.shape)      (3, 4)
```

Liste bzw. Liste von Listen in Array konvertieren

Array hat ein Attribut shape

```
print(one[0])     1

print(two[0])     [1 2 3 4]

print(one[2:])    [3 4 5 6]

print(two[:,0])   [1 5 9]
```

Zugriff über Index

Slicing auch hier, Dimensionen mittels , trennen

ND-Arrays

```
import numpy as np

one = np.zeros(5)
two = np.ones((5,5))

print(one, one.shape)
print(two, two.shape)
```

```
[0. 0. 0. 0. 0.] (5,)

[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]] (5,5)
```

Array unter Angabe von Dimensionen mit 0en oder 1en

```
print(two.ndim, two.size)
```

```
2 25
```

Neben shape gibt es auch `ndim` und `size`

Leeres Array (zufällige Werte) gegeben Dimension und Typ

```
three = np.ndarray((2,2), dtype=np.float32)
four = np.ndarray((2,2), dtype=np.int64)

print(three, three.shape)
print(four, four.shape)
```

```
[[1.67e-43 1.36e-43]
 [1.60e-43 1.54e-43]] (2, 2)

[[-9223372036854775808  2305851798912965847]
 [          4485873667  7546421577360895877]]
(2, 2)
```

Arrays um

```
import numpy as np

one = np.arange(6)
print(one, one.shape)

two = one.reshape(3,2)
print(two, two.shape)
```

```
[0 1 2 3 4 5] (6,)

[[0 1]
 [2 3]
 [4 5]] (3, 2)
```

Werte des Arrays in eine andere Form bringen

Es gilt `four[3, 0] == three[0, 3]`

Hinzufügen einer (leeren) Dimension.
Hier Achse „0“, also Werte landen in Achse 1.
Zugriff dann per `three[„0“, 3]`

```
three = np.expand_dims(one, axis=0)
print(three, three.shape)
```

```
[[0 1 2 3 4 5]] (1, 6)
```

```
four = np.expand_dims(one, axis=1)
print(four, four.shape)
```

```
[[0]
 [1]
 [2]
 [3]
 [4]
 [5]] (6, 1)
```

Hinzufügen einer (leeren) Dimension.
Hier Achse „1“, also Werte landen in Achse 0.
Zugriff dann per `four[3, „0“]`

Z nd Filter

Herausfiltern bestimmter Werte

Boolsches Array wählt die zurücktzugebenden Indizes aus.

```
import numpy as np

one = np.arange(20)
two = np.flip(one)
print(one)
print(two)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
[19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0]
```

Vergleich ergibt hier ein neues boolsches Array

```
print(one[(one % 2 == 1)])
```

```
[ 1  3  5  7  9 11 13 15 17 19]
```

```
print(one > 10)
print(two > 10)
```

```
[False False False False False False False False False False False  True True  True  True  True  True  True  True  True ]
[ True  True  True  True  True  True  True  True  True False False False False False False False False False False False]
```

```
print(np.nonzero(two > 10))
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8]),)
```

```
print(np.flatnonzero(two > 10))
```

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

```
print(one[(one > 10)])
print(one[(two > 10)])
```

```
[11 12 13 14 15 16 17 18 19]
[0 1 2 3 4 5 6 7 8]
```

Nicht die Werte, sondern die Indizes

Operationen

```
import numpy as np

a = np.array([1, 2])
b = np.array([1, 1])

print(a + b)        [2 3]

print(a * b)        [1 2]


print(a + 2)        [3 4]

print(a * 2)        [2 4]


A = np.array([[1, 2],
              [3, 4]])
B = np.array([[5, 6],
              [7, 8]])
```

Elementweise Operation

Broadcasting (Skalar wird auf Array angewandt)

```
print(A + B)    [[ 6  8]
                 [10 12]]

print(A * B)    [[ 5 12]
                 [21 32]]

print(A @ B)    [[19 22]
                 [43 50]]
```

Matrixmultiplikation

Auswahl der Achse, über die „gerechnet“ werden soll.

```
print(A.sum(), A.sum(axis=1), A.sum(axis=0))
10 [3 7] [4 6]

print(B.min(), B.max())
5 8

print(A + a)    [[2 4]
                 [4 6]]
```

Broadcasting (Array wird auf Array angewandt)

Beispiel: Formel

Scheleife über die Elemente in x und y

$$d_e(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

```python
import math

def d_e_py(x, y):
  r = 0
  for x_i, y_i in zip(x, y):
    r += (x_i - y_i)**2
  return math.sqrt(r)

print(d_e_py(
  [1, 2, 1],
  [1, 3, 2]
))
```

Ausnutzen der Operationen von NumPy, Elemente nicht selbst betrachtet!

```
1.4142135623730951
```

```python
import numpy as np

def d_e_numpy(x, y):
  return np.sqrt(
    np.power(
      (x - y),
      2
    ).sum()
  )

print(d_e_numpy(
  np.array([1, 2, 1]),
  np.array([1, 3, 2])
))
```

Beispiel: Geschwindigkeit

```
import timeit

x = np.random.rand(1000000)
y = np.random.rand(1000000)

print(timeit.timeit('d_e_numpy(x, y)', number=50, globals=globals()))
```

```
1.156090356998902 3
```

```
x_list = x.tolist()
y_list = x.tolist()

print(timeit.timeit('d_e_py(x_list, y_list)', number=50, globals=globals()))
```

```
3.586029051999503 3
```

Zeitmessung der Numpy-Funktion (Mittel über 50 Durchläufe)

Zeitmessung reines Python

Speichern und Laden

```
import numpy as np

x = np.array([[1,2],[3,4]])
y = x.T # x.transpose()

np.save('array_x', x)
x_1 = np.load('array_x.npy')
print(x_1)
```

```
[[1 2]
 [3 4]]
```

Ein einzelnes Array eine Datei speichern

(einzeln → .npy)

```
np.savetxt('array_x.txt', x)
x_2 = np.loadtxt('array_x.txt')
```

array_x.txt

```
1.0000000000000000e+00 2.000000
3.0000000000000000e+00 4.000000
```

Array als Text-datei speichern (menschenlesbar, aber langsamer und größer)

```
np.savez('arrays', x=x, y=y)
```

Mehrere Arrays in einer Datei speichern

→ .npz

```
with np.load('arrays.npz') as arrays:
  print(type(arrays))
```

```
<class 'numpy.lib.npyio.NpzFile'>
```

```
  x_3 = arrays["x"]
  y_3 = arrays["y"]

print(x_3, y_3)
```

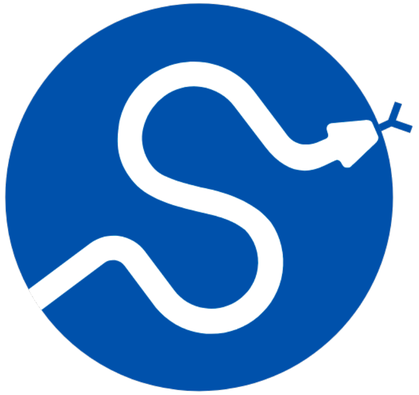
```
[[1 2]
 [3 4]]
[[1 3]
 [2 4]]
```

Öffnen einer .npz erzeugt eine Art Wörterbuch für die hinzugefügten Arrays

II.
Wissenschaftliches Rechnen

2. SciPy

Installation



- Python Paket
  - https://scipy.org/
- Installation z.B. mit `pip3 install scipy`
- Import

```
import scipy
```

Warum SciPy?

- Sammlung wichtiger mathematische Funktionen
- Erweiterung von NumPy (nutzt NumPy intern)
  - NumPy bietet Basis (Matrizen & Datenstrukturen)
  - SciPy bietet Algorithmen für Anwendungen

SciPy Teilpakete

- Fourier Transformationen `scipy.fft`
- Dünnbesetzte Matrizen `scipy.sparse`
- Distanzfunktionen `scipy.spatial.distance`
- Lineare Algebra `scipy.linalg`
- Statistische Funktionen `scipy.stats`

Ähnlich zu `numpy.linalg`

ünnbesetzte Matrizen

Jede 0 wird als Zahl abgespeichert

$$I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Verschiedene Formate je nach Aufteilung der 0en und der benötigten Funktionen.

```
from scipy.sparse import (
  bsr_array, coo_array,
  csc_array, csr_array,
  dia_array, dok_array,
  lil_array
)
```

Nur die von 0 verschiedenen Werte und deren Position speichern

```
import numpy as np

i_20_n = np.identity(20, dtype=np.int64)

print("Number of elements",
  i_20_n.size)                          400
print("Size of element",
  i_20_n.itemsize)                        8
print("Bytes used",
  i_20_n.nbytes)                       3200
```

400 * 8 = 3200

```
from scipy.sparse import identity
i_20_s = identity(20, dtype=np.int64, format='dia')

print("Number of elements",
  i_20_s.data.size + i_20_s.offsets.size)          21
print("Size of element",
  i_20_s.data.itemsize, i_20_s.offsets.itemsize)  8 4
print("Bytes used",
  i_20_s.data.nbytes + i_20_s.offsets.nbytes)     164
```

20 * 8 + 1 * 4 = 164

https://docs.scipy.org/doc/scipy/reference/sparse.html

Statistische Funktionen

Beispiel: Binomialverteilung

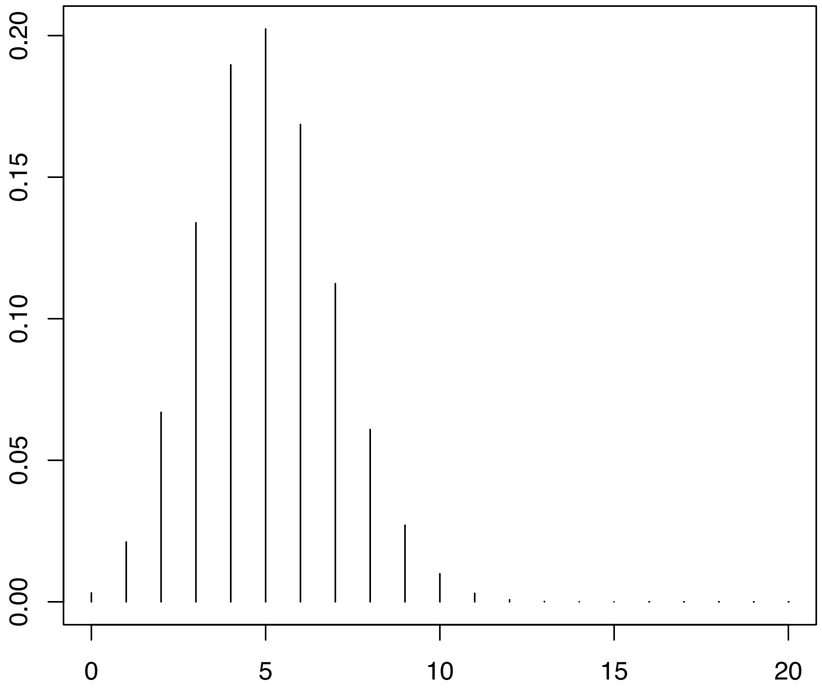
$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

```
from scipy.stats import binom

n, p = 20, 0.25
mean, variance = binom.stats(n, p, moments='mv')

print(mean, variance)
```

```
5.0 3.75
```



https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.binom.html

Nochmal: Beispiel Geschwindigkeit

Eine eigene Implementierung mittels NumPy ist zwar schon schneller, aber SciPy bietet eine besser optimierte Implementierung.

```
import timeit
x = np.random.rand(1000000)
y = np.random.rand(1000000)

print(timeit.timeit('d_e_numpy(x, y)', number=50, globals=globals()))
```

```
1.156903569989023
```

```
x_list = x.tolist()
y_list = x.tolist()

print(timeit.timeit('d_e_py(x_list, y_list)', number=50, globals=globals()))
```

```
3.586029051995033
```

```
from scipy.spatial import distance

print(timeit.timeit('distance.euclidean(x, y)', number=50, globals=globals()))
```

```
0.0945173029952033
```

Nutzt intern wiederum `numpy.linalg.norm`

cs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.euclidean.html

Zusammenfassung

I. Projektaufgabe 3

    1. Herangehensweise & Tipps

II. Wissenschaftliches Rechnen

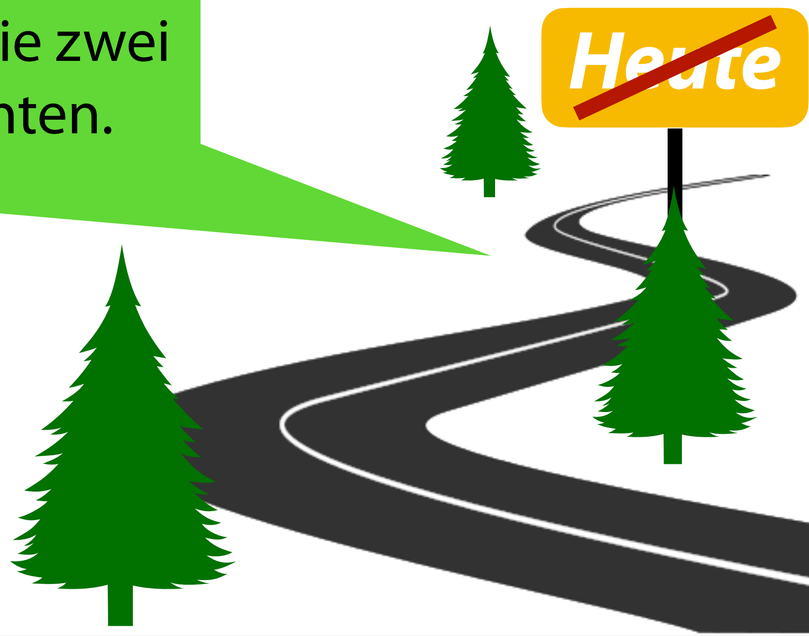
    1. NumPy

    2. SciPy

III. Reguläre Ausdrücke

Aufgabe 3 bleibt über Weihnachten freigeschaltet.

Die Bearbeitungszeit ist also drei Wochen ohne die zwei Wochen Weihnachten.



Inhaltsübersicht

1. Programmiersprache Python
   a) *Einführung, Erste Schritte*
   b) *Grundlagen*
   c) *Fortgeschritten*
2. Auszeichnungssprachen
   a) *LaTeX, Markdown*
3. Benutzeroberflächen und Entwicklungsumgebungen
   a) *Jupyter Notebooks lokal und in der Cloud (Google Colab)*
4. Versionsverwaltung
   a) *Git, GitHub*
5. Wissenschaftliches Rechnen
   a) *NumPy, SciPy*
6. Datenverarbeitung und -visualisierung
   a) Pandas, matplotlib, NLTK
7. Machine Learning (scikit-learn)
   a) Grundlegende Ansätze (Datensätze, Auswertung)
   b) Einfache Verfahren (Clustering, …)
8. DeepLearning
   a) TensorFlow, PyTorch, HuggingFace Transformers