

# TSRI-2

June 21, 2024

## 1 Übung 2.1

Gruppenname: TSRI

- Christian Rene Thelen @cortex359
- Leonard Schiel @leo\_paticumbum
- Marine Raimbault @Marine Raimbault
- Alexander Ivanets @sandrium

### 1.1 2.1.1 Shapes und Co.

Bei vielen Data Science Projekten werden Daten zunächst zugänglich gemacht, sodass sie als Numpy Arrays vorliegen. Das Verständnis von *Shapes* solcher Arrays ist eine wichtige Voraussetzung für eine produktive Arbeit mit diesen Datenstrukturen.

Nehmen Sie die Folien zur heutigen Vorlesung zur Hand.

#### Ihre Aufgaben

- (1) Betrachten Sie den untenstehenden Code. Beantworten Sie: Welchen [Shape](#) hat das Numpy Array `a`?

```
[1]: import numpy as np
a = np.array([5, 10, 15, 20])
```

```
[2]: print(f"Die Shape von `a` ist {a.shape}")
```

Die Shape von ``a`` ist (4,)

- (2) Lässt sich das Array `a` als (mathematischen) Vektor interpretieren? Wenn `a` ein Vektor ist, können Sie ihn [transponieren](#)?

In der Mathematik ist ein Vektor ein Element eines Vektorraums und ein Vektorraum eine algebraische Struktur, in der die Vektorraumaxiome erfüllt sind. Das NumPy Array `a` lässt sich als Vektor in einem Vektorraum über den Körper  $K$  interpretieren, wobei Addition `+` und Multiplikation `*` in  $K$  sowie die Vektoraddition `+` und die Skalarmultiplikation `*` sich wie zu erwarten verhalten.

Auch das euklidische Skalarprodukt mit `np.dot` und eine **elementweise** Vektor-Vektor-Multiplikation mit `*` sind definiert, jedoch lässt sich `a` nicht transponieren, da es sich um ein 1-dimensionales Array der Shape (4,0) handelt und die Unterscheidung zwischen Zeilen und Spalten bei nur einer Dimension nicht sinnvoll ist. Zur Transposition ist daher eine Umwandlung

in ein (zumindest) 2-dimensionales Array der Shape (4,1) notwendig. Dies lässt sich z.B. mit `a.reshape((4,1))` bewerkstelligen, aber besser noch so:

```
[3]: a_vec = a[:, np.newaxis]
      print(f"{a_vec.shape=} und {a_vec.T.shape=}")
```

`a_vec.shape=(4, 1)` und `a_vec.T.shape=(1, 4)`

- (3) Betrachten Sie das unten stehende Array `b`. Erzeugen Sie aus `b` bitte zwei neue Arrays `c` und `d`: `c` soll ein Spaltenvektor sein, und `d` soll ein Zeilenvektor sein. Wie gehen Sie vor?

```
[4]: b = np.array([2, 4, 6, 8, 10])
      c, d = b[np.newaxis, :], b[:, np.newaxis]
```

Lässt sich `c` bzw. `d` transponieren? Warum?

Ja, weil es sich um 2-dimensionale Arrays handelt.

- (4) Betrachten Sie die dritte Zeile des unten stehenden Codes: Hat sich durch die Zeile `k[0, 1] = 10` das Array `m` geändert? Falls ja, warum? Falls nein, warum hat es sich nicht geändert?

```
[5]: k = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
      m = k[:2, 1:3]
      k[0, 1] = 10
```

Ja, denn `m` ist eine Referenz (genannt *view*) auf einen Ausschnitt von `k` und das Element `k[0, 1]` ist in dem Ausschnitt unter der Referenz `m[0, 0]` enthalten. (basic indexing)

- (5) Betrachten Sie die dritte Zeile des unten stehenden Codes: Hat sich durch die Änderung des Arrays `p` der Inhalt von `q` geändert? Falls ja, warum? Falls nein, warum hat es sich nicht geändert?

```
[6]: p = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
      q = p[[1, 2], 0]
      p[1, 0] = 25
```

Nein, der Inhalt von `q` hat sich nicht geändert, da beim *advanced indexing* immer eine Kopie der Daten erzeugt wird.

## 1.2 2.1.2 Broadcasting - Teil 1

Nehmen Sie sich die Folien zur heutigen Vorlesung zur Hand. Unten sehen Sie eine Operation mit zwei Numpy Arrays `a` und `b`.

### Ihre Aufgaben

- (1) Betrachten Sie die unten stehende Code-Zelle. Welchen Shape (Anzahl Zeilen, Anzahl Spalten) wird das Numpy Array `c` erhalten?
- Tragen Sie ihre Voraussage in die Variable `shape` ein.
  - Begründen Sie Ihre Vorhersage mit den drei Broadcasting Regeln aus der Vorlesung.
  - Berechnen Sie auf Papier das zu erwartende Ergebnis. Tragen Sie es als Numpy Array in die Variable `ergebnis` ein.
  - Führen Sie dann jeweils die Zelle aus, um zu schauen, ob Sie richtig lagen.

```
[7]: import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

c = a + b

# Ihre Vorhersagen
shape = (3,)
ergebnis = np.array([5, 7, 9])

print('{}'.format('Shape korrekt.' if np.all(shape==c.shape) else 'Shape nicht_
↳korrekt.'))
print('{}'.format('Ergebnis korrekt.' if np.array_equal(ergebnis, c) else_
↳'Ergebnis nicht korrekt.'))
```

Shape korrekt.

Ergebnis korrekt.

- (2) Betrachten Sie die unten stehende Code-Zelle. Welchen Shape (Anzahl Zeilen, Anzahl Spalten) wird das Numpy Array c erhalten?

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

c = a + b

shape = (3, 3) # Ihre Vorhersage
ergebnis = np.array([
    [5, 6, 7],
    [6, 7, 8],
    [7, 8, 9],
])

print('{}'.format('Shape Korrekt.' if np.all(shape==c.shape) else 'Shape nicht_
↳korrekt.'))
print('{}'.format('Ergebnis korrekt.' if np.array_equal(ergebnis, c) else_
↳'Ergebnis nicht korrekt.'))
```

Shape Korrekt.

Ergebnis korrekt.

- (3) Betrachten Sie die unten stehende Code-Zelle. Welchen Shape (Anzahl Zeilen, Anzahl Spalten) wird das Numpy Array c erhalten?

```
[9]: a = np.array([[1, 0, 0], [0, 0, 2]])
b = np.ones(12).reshape((4, 1, 3))

c = a * b
```

```

shape = (4, 2, 3) # Ihre Vorhersage

# Regel 1 (padding with ones on left side): a(2,3) -> a(1,2,3)
a_1 = np.array([
    [
        [1, 0, 0], [0, 0, 2]
    ]
])
print(f"a {a.shape} -> {a_1.shape}")

# Regel 2 (stretching from 1 to match the other shape) a(1,2,3) -> a(4,2,3)
a_2 = np.array([
    [[1, 0, 0], [0, 0, 2]],
    [[1, 0, 0], [0, 0, 2]],
    [[1, 0, 0], [0, 0, 2]],
    [[1, 0, 0], [0, 0, 2]]
])
print(f"a {a_1.shape} -> {a_2.shape}")

# Regel 2 (stretching from 1 to match the other shape) b(4,1,3) -> b(4,2,3)
b_1 = np.array([
    [[1, 1, 1], [1, 1, 1]],
    [[1, 1, 1], [1, 1, 1]],
    [[1, 1, 1], [1, 1, 1]],
    [[1, 1, 1], [1, 1, 1]]
])
print(f"b {b.shape} -> {b_1.shape}")

# Elementweise Multiplikation.
ergebnis = a_2

print('{}'.format('Shape korrekt.' if np.all(shape==c.shape) else 'Shape nicht_
↳korrekt.'))
print('{}'.format('Ergebnis korrekt.' if np.array_equal(ergebnis, c) else_
↳'Ergebnis nicht korrekt.'))

```

```

a (2, 3) -> (1, 2, 3)
a (1, 2, 3) -> (4, 2, 3)
b (4, 1, 3) -> (4, 2, 3)
Shape korrekt.
Ergebnis korrekt.

```

### 1.3 2.1.3 Broadcasting Teil 2

Ein häufiger Vorverarbeitungsschritt, der Ihnen in verschiedenen Data Science Projekten begegnet wird, ist die Normierung von Daten. Häufig werden dabei Datensätze, z.B. Zeitreihen, auf Mittelwert 0 und Standardabweichung 1 normiert. Dieser Vorgang wird auch *z-scoring* (oder:

Standardisierung) genannt:

Sei  $x_i$  der  $i$ -te Datenpunkt einer Zeitreihe. Dann ist der z-score dieses Datenpunkts definiert als

$$\tilde{x}_i = \frac{x_i - \bar{x}}{\sigma},$$

wobei  $\bar{x}$  der Mittelwert sowie  $\sigma$  die Standardabweichung der Zeitreihe bezeichnen.

Das in der unten stehenden Code-Zelle definierte Numpy Array X enthält 5 Zeitreihen (Spalten) mit je 30 Einträgen (Zeilen).

- Nutzen Sie Broadcasting sowie `np.mean` und `np.std`, um die Zeitreihen auf Mittelwert 0 und Varianz 1 zu normieren. Die transformierten Zeitreihen sollen im Numpy Array Y gespeichert werden. Dabei schreiben Sie nur eine Zeile Code, um die Zeitreihen zu transformieren.
- Beachten Sie das `axis` Argument bei `np.mean` und `np.std`.

```
[10]: import numpy as np
      np.random.seed(0)

      X = np.random.random((30, 5))

      # Ihr Ergebnis:
      assert np.mean(X, axis=0).shape == (5,), "5 Zeitreihen sollten 5 Mittelwerte_
      ↪haben"
      Y = (X - np.mean(X, axis=0)) / np.std(X, axis=0)
```

#### 1.4 2.1.4 Advanced Indexing

Nehmen Sie sich die Folien der heutigen Vorlesung zur Hand. In diesem Übungsteil geht es darum, Sie mit *Advanced Indexing* in Numpy Arrays vertraut zu machen.

##### Ihre Aufgaben

- (1) Betrachten Sie den unten stehenden Code-Zellen.
  - Tragen Sie in der Variable `vorhersage` ein, welches Array Sie in Variable `y` erwarten.
  - Prüfen Sie durch Ausführen des Codes, ob Ihre Vorhersage richtig war.
  - Beantworten Sie die Frage: Wodurch wird der Shape des Arrays `y` bestimmt?

```
[11]: import numpy as np

      x = np.array([[1, 2], [3, 4], [5, 6]])
      y = x[[0,1,2], [0,1,0]]

      vorhersage = np.array([1, 4, 5]) # Ihre Vorhersage
      print('{}'.format('Ergebnis korrekt.' if np.array_equal(vorhersage, y) else_
      ↪'Ergebnis nicht korrekt.'))
```

Ergebnis korrekt.

- (2) Betrachten Sie den unten stehenden Code-Zellen.

```
[12]: x = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])

rows = np.array([[0,0],[3,3]])
cols = np.array([[0,2],[0,2]])
y = x[rows,cols]

vorhersage = np.array([
    [0, 2], [9, 11]
]) # Ihre Vorhersage

print('{}'.format('Ergebnis korrekt.' if np.array_equal(vorhersage, y) else
    ↪'Ergebnis nicht korrekt.'))
```

Ergebnis korrekt.

(3) Betrachten Sie den unten stehenden Code-Zellen.

```
[13]: x = np.array([[0, 1, 2],[3, 4, 5],[6, 7, 8],[9, 10, 11]])

y = x[1:4,[1,2]] # Kombination von Slicing und Advanced Indexing

vorhersage = np.array([
    [4, 5],
    [7, 8],
    [10, 11]
]) # Ihre Vorhersage

print('{}'.format('Ergebnis korrekt.' if np.array_equal(vorhersage, y) else
    ↪'Ergebnis nicht korrekt.'))
```

Ergebnis korrekt.