Vektoren, Matrizen, Tensoren

Inhalt

In diesem Modul schauen wir uns an, wie wir mit Python mathematische Berechnungen durchführen, die für Data Science von Bedeutung sind. Dafür benutzen wir die Bibliothek NumPy, die vor allem mathematische Funktionen bietet, aber auch einen eigenen Datentyp mitbringt, das Objekt `ndarray`, das sich besonders für die Berechnung von Vektoren und Matrizen eignet.

Listen können als Vektoren (die man aus der Linearen Algebra kennt) und Tabellen als Matrizen dargestellt werden. Oft steht man bei der Datenauswertung vor der Aufgabe, Gleichungen mit Vektoren oder Matrizen zu lösen. Hierbei kommen die mathematischen Verfahren der Vektor- und Matrizenrechnung zum Einsatz.

Für diese und weitere Berechnungen bietet die Python-Bibliothek NumPy passende Datentypen und Funktionen an. NumPy steht für Numerical Python und ist die verbreitetste Python-Bibliothek für die Arbeit mit numerischen Daten u.a. in Wissenschaft und Technik. Sie bildet auch die Basis für weitere Bibliotheken, wie Pandas und Matplotlib, die selbst NumPy-Funktionen verwenden. Deshalb sind manche NumPy-Funktionen auch dort vorhanden.

Um NumPy zu verwenden, musst du in Jupyter Notebook am Anfang deines Codes (bevor du die erste Funktion verwendest) die Bibliothek importieren. Damit du dir beim späteren Verwenden der NumPy-Datenobjekte Tipparbeit sparst, ist es üblich, diese als `np` abzukürzen.

```
import numpy as np
```

Das zentrale Datenobjekt in NumPy ist das `ndarray`. Der Name steht für n-dimensionales Array, also ein Datenobjekt, das je nach Bedarf Zeilen, Spalten oder weitere Achsen (Dimensionen) haben kann. Je nach Anzahl der Dimensionen bezeichnet man diese Datenobjekte in der Mathematik wie folgt:

- 1 Dimension: Vektor
- 2 Dimensionen: Matrix
- 3 Dimensionen: Tensor

ndarray ist dabei lediglich der Klassenname für dieses n-dimensionale Datenobjekt. Die Methode zum Anlegen eines solchen Objekts lautet np.array( ).

Du kannst ein ndarray wie folgt anlegen:

```
# Vektor
a = np.array([1, 2, 3])

# 2x2-Matrix
b = np.array([[1, 2], [3, 4]])

# 2x2x2-Tensor
c = np.array([[[1,2],[3,4]], [[5,6],[7,8]]])
```

oder indem du eine vorhandene Liste in ein ndarray umwandelst:

```
a = [[1,2],[3,4]]
a = np.array(a)
print(a)
```

Ergebnis:

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

Beachte, dass ndarray in Matrixform und ohne Kommas angezeigt wird.

ndarray ist deutlich schneller und speichersparender als Python-Listen und deshalb für große Datenmengen geeignet. Einschränkungen sind, dass alle Elemente vom selben Datentyp sein müssen und die Größe der Liste unveränderlich ist. Es gibt also keinen append()-Befehl.

Deswegen legt man ein ndarray mit der nötigen Größe an und füllt es durch eine der folgenden Funktionen zero(), ones(), usw. mit Anfangswerten, die später überschrieben werden:

```
# 2x3-Matrix anlegen und füllen:
np.zeros([2,3]) # füllt mit 0.
np.ones([2,3]) # füllt mit 1.
np.full([2,3],7) # füllt mit 7.
np.random.random([2,3]) # füllt mit Zufallszahlen
```

Die Elemente sind standardmäßig float, außer man legt sie durch den Parameter dtype anders fest, z.B. als int:

```
np.zeros([2,3], dtype=int) # füllt mit 0
```

ndarray

Der Vorteil von `ndarray` ist, dass wir damit genau so rechnen können, wie man es von der Vektor- und Matrizenrechnung kennt.

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

# Vektor vervielfachen
print(2*a)                                  # Ergebnis: [2 4]

# Vektoren addieren
print(a+b)                                  # Ergebnis: [4 6]

# elementweise Multiplikation (1*3, 2*4)
print(a*b)                                  # Ergebnis: [3 8]

# Skalarprodukt berechnen (1*3 + 2*4)
print(a@b)                                  # Ergebnis: 11
```

Die Rechenoperationen mit Matrizen und Tensoren funktionieren genauso wie mit Vektoren. Darüber hinaus gibt es für quadratische Matrizen die Berechnung der Determinante:

```
A = np.array([[1,2],[3,4]])
# Determinante berechnen:
print(np.linalg.det(A))
```

Ergebnis: `-2.`

und das Invertieren der Matrix:

```
Ainv = np.linalg.inv(A)
print(Ainv)
```

Ergebnis:
```
[[-2. 1. ]
[ 1.5 -0.5]]
```

Beachte, dass eine Matrix nur invertiert werden kann, wenn sie quadratisch ist und ihre Determinante ungleich Null ist. Wir werden das in der nächsten Einheit zum Lösen von Gleichungssystemen verwenden, mit denen man z.B. Produktionsprozesse beschreiben kann.