Python - Programmierung für die Computerbasierte Intelligenz

By Tobias Häuser & Chris Gerlach

Vorlesung 3

25.08.2018

1. Matplotlib
    1. Implementierung von Bibliotheken
    2. Visualisierung Einführung

2. k-Nearest Neighbors (kNN)
    1. Eigeneimplementierung
    2. Sklearn Beispiel

1.1. Implementierung von Bibliotheken

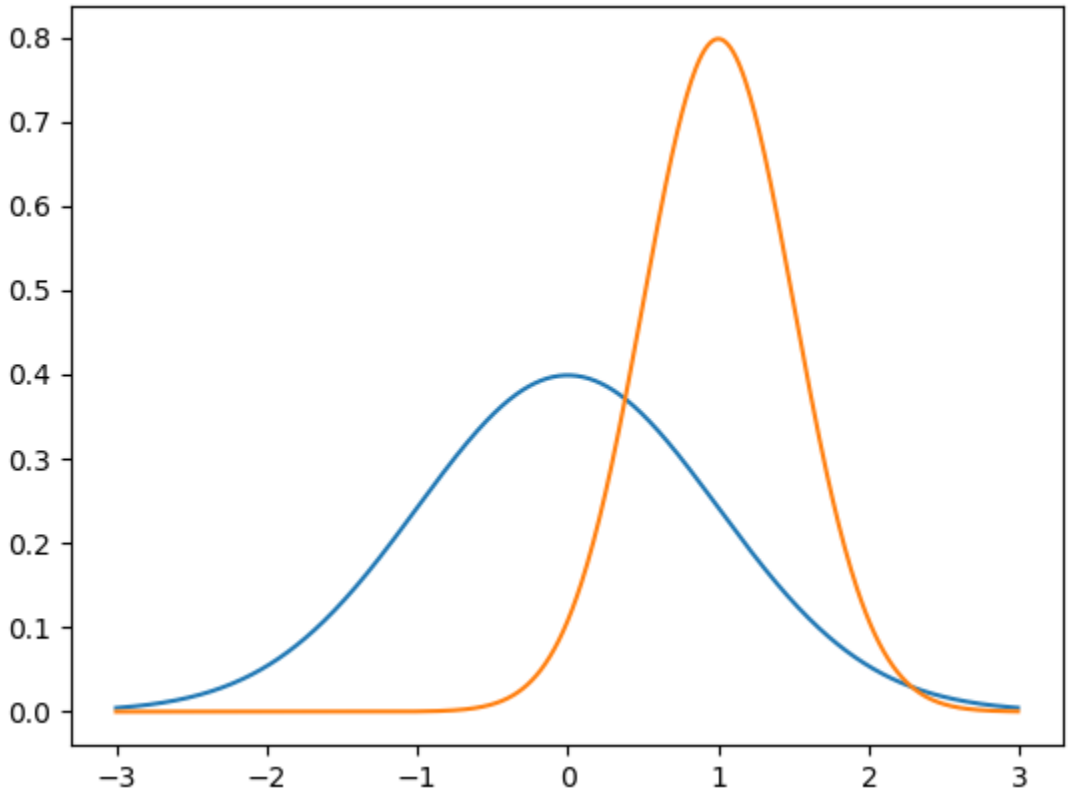
- Hinzufügen von Bibliotheken im Projekt Interpreter
- Strg + Alt +S -> Projekt Interpreter
  - Auflistung aller Libarys
  - + -> Hinzufügen
  - - -> Entfernen
  - ↑ -> Updaten

- Hinzufügen, einfach in der Suchleiste die gewünschte Bibliotheken suchen & auf installieren klicken

1.2. Visualisierung Einführung

- Zeichnen einer Linie

```
from scipy.stats import norm
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-3, 3, 0.01)

plt.plot(x, norm.pdf(x))
plt.plot(x, norm.pdf(x, 1.0, 0.5))
plt.show()
# Grifik als png speichern
# plt.savefig('grafik.png', format='png')
```



1.2. Visualisierung Einführung

- Achsen anpassen

```
axes = plt.axes()
# Achsenbereich angeben
axes.set_xlim([-5, 5])
axes.set_ylim([0, 1.0])

# Achsenpunkte angeben
axes.set_xticks([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5])
axes.set_yticks([0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])

# Grid anzeigen
axes.grid()
```

1.2. Visualisierung Einführung

- Linien anpassen

```
# Linientyp anpassen
plt.plot(x, norm.pdf(x), 'm-')
plt.plot(x, norm.pdf(x, 1.0, 0.5), 'r:')
```

Verfügbare Farben:

- b: blue
- g: green
- r: red
- c: cyan
- m: magenta
- y: yellow
- k: black
- w: white

Linienformen:

- 'r:' = rot gepunktet
- 'r-' = rot durchgezogen
- 'r--' = rot gestrichelt
- 'r-.' = rot strich / punkt abwechselnd

1.2. Visualisierung Einführung

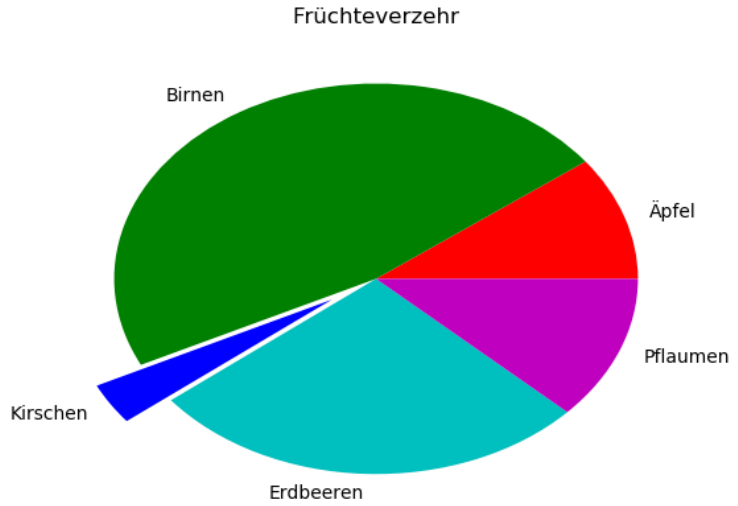
- Achsen- & Legendenbeschriftung

```
# Achsen & Legendenbeschriftung
plt.xlabel('Anzahl')
plt.ylabel('Wahrscheinlichkeit')
plt.legend(['Äpfel', 'Birne'], loc=4)
```

1.2. Visualisierung Einführung

- Tortendiagramm

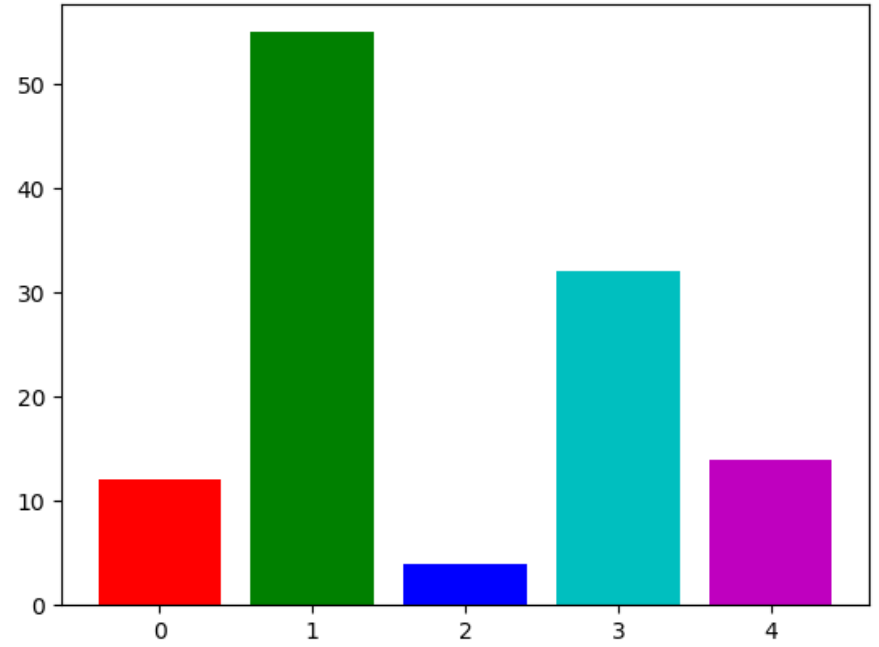
```
# Tortendiagramm
values = [12, 55, 4, 32, 14]
colors = ['r', 'g', 'b', 'c', 'm']
explode = [0, 0, 0.2, 0, 0]
labels = ['Äpfel', 'Birnen', 'Kirschen', 'Erdbeeren', 'Pflaumen']
plt.pie(values, colors= colors, labels=labels, explode = explode)
plt.title('Früchteverzehr')
plt.show()
```



1.2. Visualisierung Einführung

- Balkendiagramm

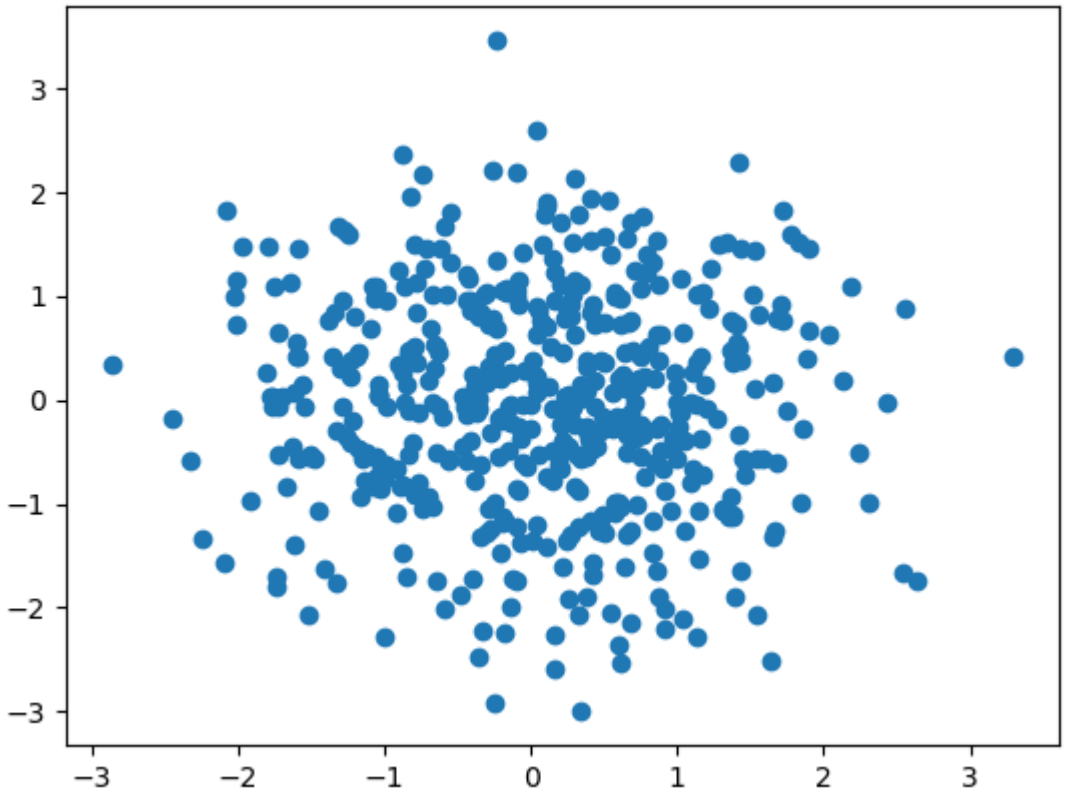
```
# Balkendiagramm
values = [12, 55, 4, 32, 14]
colors = ['r', 'g', 'b', 'c', 'm']
plt.bar(range(0,5), values, color= colors)
plt.show()
```



1.2. Visualisierung Einführung

- Punktediagramm

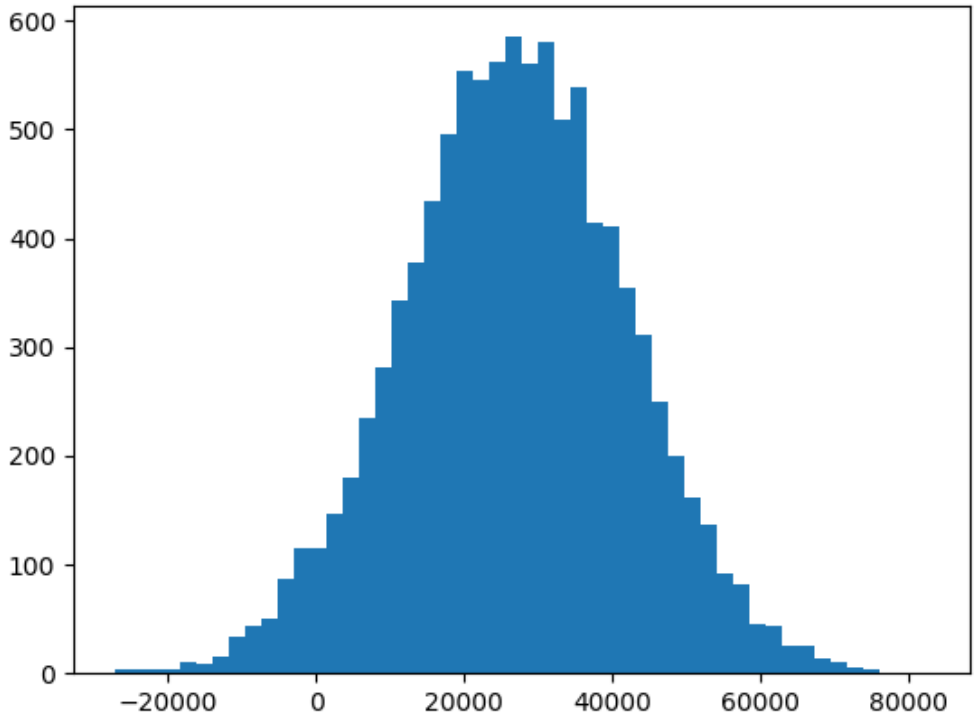
```
# Punktdiagramm
from pylab import randn

X = randn(500)
Y = randn(500)
plt.scatter(X,Y)
plt.show()
```



1.2. Visualisierung Einführung

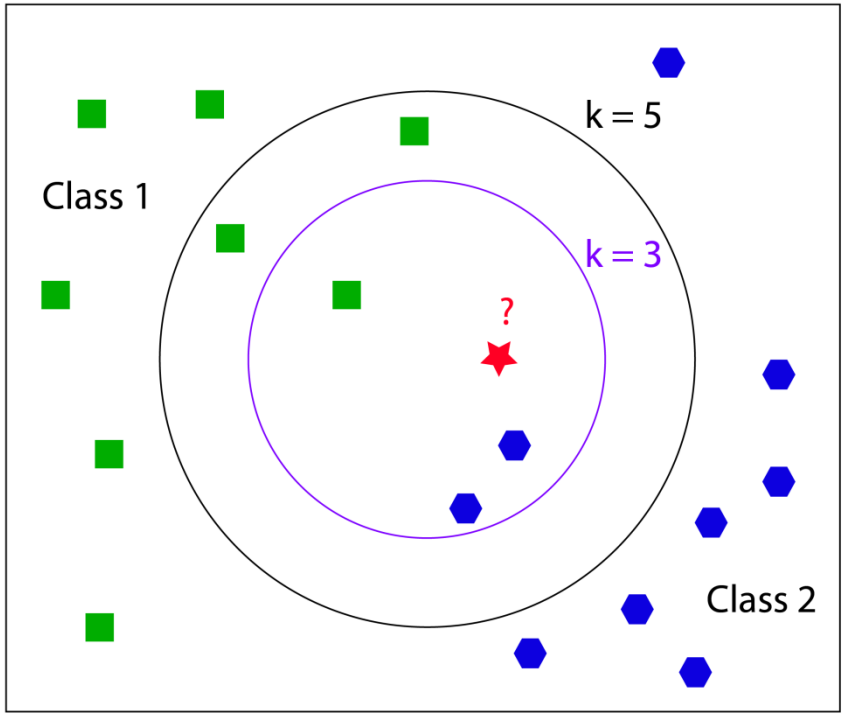
- Histogramm

```
# Histogramm
incomes = np.random.normal(27000, 15000, 10000)
plt.hist(incomes, 50)
plt.show()
```



2. k-Nearest Neighbors (kNN)

- ist ein Klassifikationsverfahren, bei dem eine Klassenzuordnung unter Berücksichtigung seiner k nächsten Nachbarn vorgenommen wird
  - Zuordnung eines Datensatzes zu einer Klasse, anhand seiner k nächsten Nachbarn und deren Klasse



https://www.quora.com/How-is-the-k-nearest-neighbor-algorithm-different-from-k-means-clustering

2. k-Nearest Neighbors (kNN)

- die Bestimmung der nächsten Nachbarn erfolgt über euklidische Distanzberechnung

$$\text{Distanz} = \sqrt{\sum_{i=1}^{n}(xi - yi)^2} = \sqrt{(x1 - y1)^2 + (x2 - y2)^2}$$

2.1. Eigeneimplementierung

- Erstellung einer Klasse ‚knn‘
- Benötigt werder folgende Bibliotheken
  - operator
  - Numpy
- Zwei Funktionen sind zu erstellen

2.1. Eigeneimplementierung

- Bibliotheken & erste Funktion

```
import operator
import numpy as np


def erstelleDaten():
    # Beispiel Daten als Vorgabe
    gruppe = np.array([[1.0, 1.1], [1.0, 1.0], [0, 0], [0, 0.1]])
    klassen = ['A', 'A', 'B', 'B']
    return gruppe, klassen
```

2.1. Eigeneimplementierung

- zweite Funktion

```
def klassifizierung(inPunkt, datenSet, klassen, k):
    datenSetSize = datenSet.shape[0]

    # Differenz der Gruppe zum Datensatz | rückwärts durchgehend
    # Quadrieren und Wurzel für euklidische Distanz
    # tile Wiederholt Elemente eines Array (Array, Anzahl)
    diffMatrix      = np.tile(inPunkt, (datenSetSize, 1)) - datenSet
    quadDiffMat     = diffMatrix**2

    # Berechnung der gesamt Distanz (x+y Distanz)
    quadDistanzen   = quadDiffMat.sum(axis=1)
    distanzen       = quadDistanzen**0.5
    sortDistIdx     = distanzen.argsort()
```

2.1. Eigeneimplementierung

```
# leeres Dicto
klassenZaehler={}

# zaehlen der k Klassen, anhand der geringsten Entfernung
for i in range(k):
    klasseI = klassen[sortDistIdx[i]]
    klassenZaehler[klasseI] = klassenZaehler.get(klasseI, 0) + 1

# Rückwärtssortierung
# Key = nach welcher Position sortiert wird | revers = Rückwärts
sortedClassCount = sorted(klassenZaehler.items(), key=operator.itemgetter(1), reverse=True)

return sortedClassCount[0][0]
```

2.1. Eigeneimplementierung

- Weiteres Beispiel, kürzere Funktion

```
def klassi_kurz(inPunkt, datenSet, klassen, k):
    distanzen = np.array([])

    # Distanzen berechnen
    for item in datenSet:
        euklidDist = ((inPunkt[0] - item[0])**2 + (inPunkt[1] - item[1])**2)**0.5
        distanzen = np.append(distanzen, euklidDist)

    # Sortierung von kleinster zur größten Distanz
    sortDistIdx = distanzen.argsort()

    # leeres Dicto
    klassenZaehler = {}
```

2.1. Eigeneimplementierung

```
# zaehlen der k Klassen, anhand der geringsten Entfernung
for i in range(k):
    klasseI = klassen[sortDistIdx[i]]
    klassenZaehler[klasseI] = klassenZaehler.get(klasseI, 0) + 1

# Rückwärtssortierung
# Key = nach welcher Position sortiert wird | revers = Rückwärts
sortedClassCount = sorted(klassenZaehler.items(), key=operator.itemgetter(1), reverse=True)

return sortedClassCount[0][0]
```

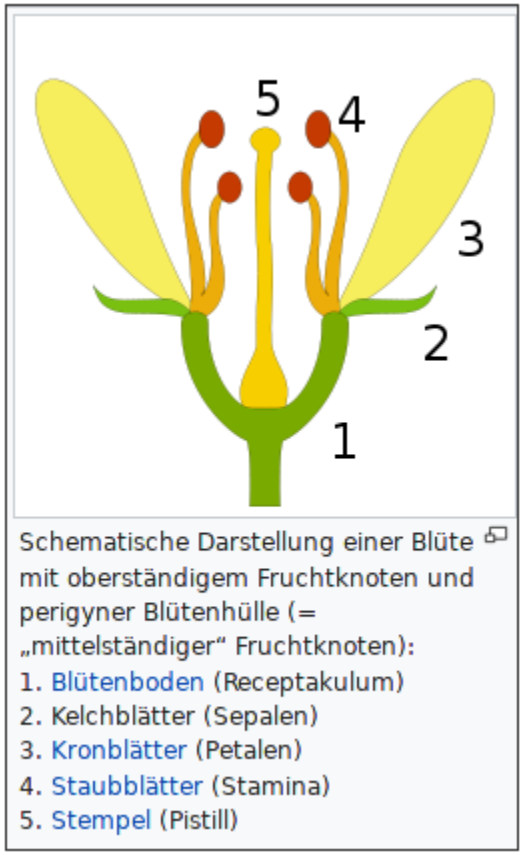
2.1. Eigeneimplementierung

- Aufruf von kNN
  - Ausgabe: `Ergebnis: B`

```
import KNN

# Vorgabe Daten beziehen
group, labels = KNN.erstelleDaten()

# Aufruf kNN & Übergabe des zu Bestimmenden Punktes,
# anhand von k Neighbors
output = KNN.klassifizierung([0.2, 0.5], group, labels, 3)
print('Ergebnis: ' + output)
```

2.2. Sklearn Beispiel

- Beispiel von einer Webseite, bezogen auf die Kronblätterlänge und dessen Klassifizierung



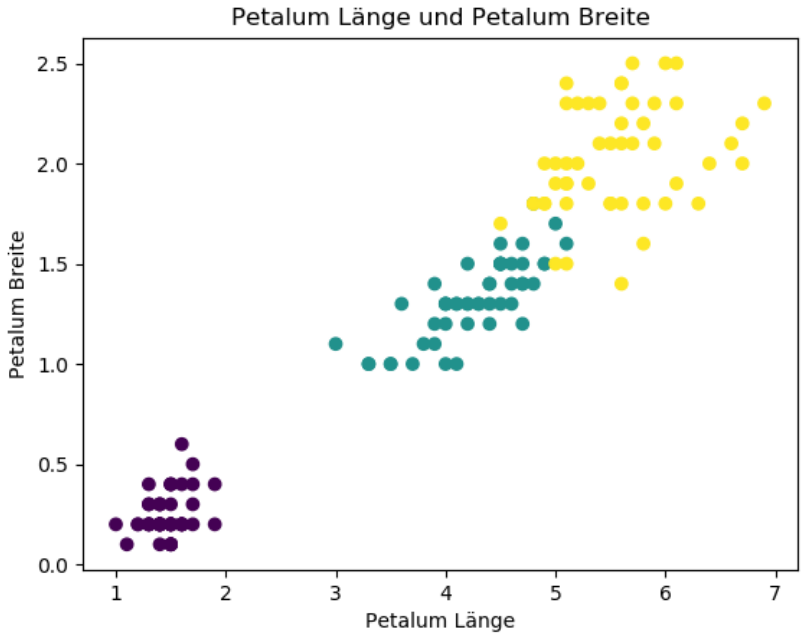
Schematische Darstellung einer Blüte mit oberständigem Fruchtknoten und perigyner Blütenhülle (= „mittelständiger“ Fruchtknoten):
1. Blütenboden (Receptakulum)
2. Kelchblätter (Sepalen)
3. Kronblätter (Petalen)
4. Staubblätter (Stamina)
5. Stempel (Pistill)

2.2. Sklearn Beispiel

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# Datensatz laden
iris = load_iris()
```

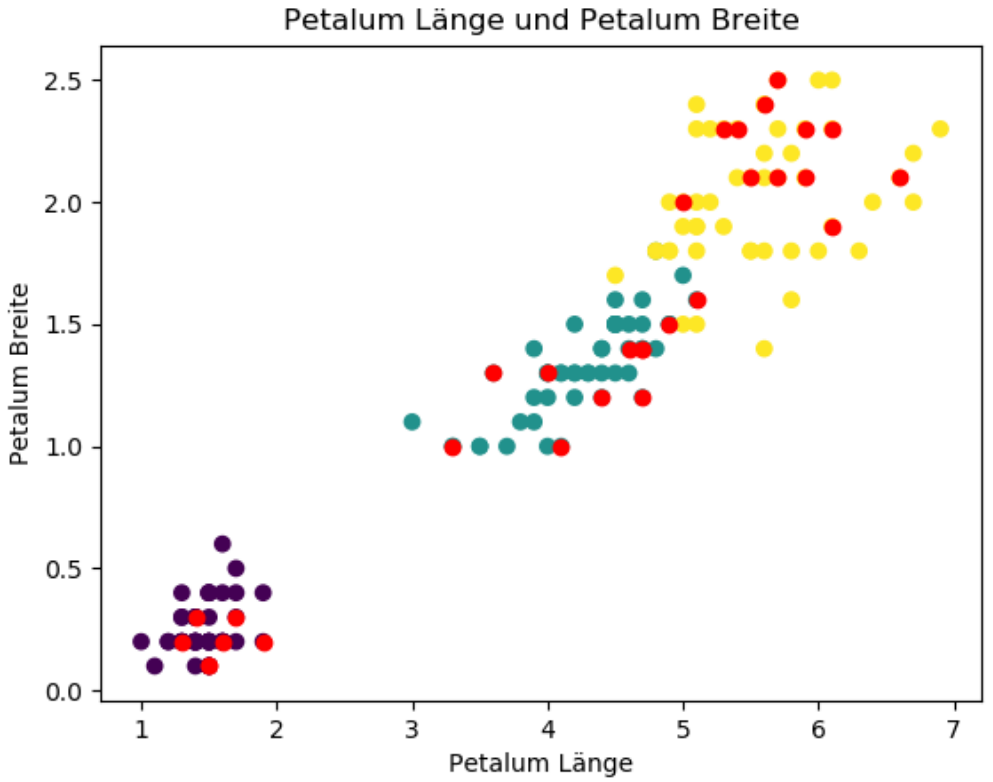
2.2. Sklearn Beispiel

```
# Daten plotten
colors = list(iris['target'])

plt.scatter(iris['data'][:, 2], iris['data'][:, 3], c=colors)
plt.title("Petalum Länge und Petalum Breite")
plt.xlabel("Petalum Länge")
plt.ylabel("Petalum Breite")
plt.show()
```



2.2. Sklearn Beispiel

```
# Trainings- und Testdaten erzeugen
x_train, x_test, y_train, y_test = train_test_split(
                    np.array(iris['data']),
                    np.array(iris['target']),
                    test_size=0.2)
```



2.2. Sklearn Beispiel

```
## kNN Algorithmus
knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(x_train, y_train)

pred = knn.predict(x_test)

print(pred)
```

Quellen

- https://blog.ancud.de/home/-/blogs/einfuhrung-in-machine-learning-mit-python-k-nearest-neighbours?redirect=%2Fhome%3Fp_p_id%3D33%26p_p_lifecycle%3D0%26p_p_state%3Dnormal%26p_p_mode%3Dview%26p_p_col_id%3Dcolumn-1%26p_p_col_count%3D1%26p_r_p_564233524_tag%3Dpython