Technische Universität Braunschweig

Medizinische Hochschule Hannover

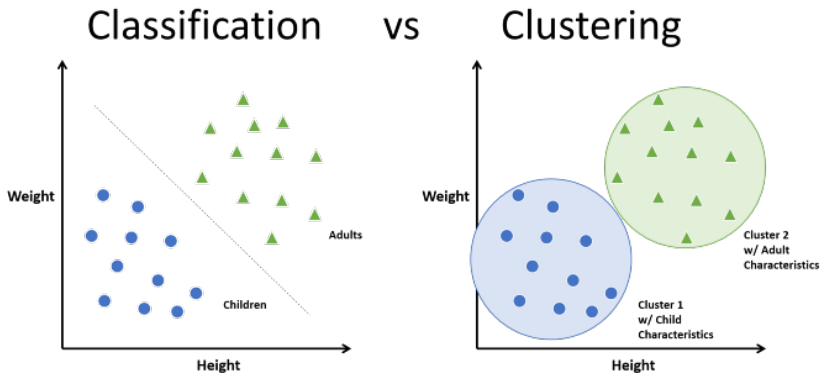
PETER L. REICHERTZ INSTITUT FÜR MEDIZINISCHE INFORMATIK

VL 3 (Teil1) – Klassifikation / Clusteranalyse

Mareike Schulze

Klassifikation vs. Clusteranalyse

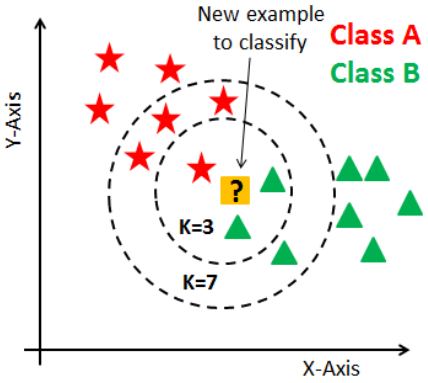
- Klassifikation:
  - Zuordnung von Datenpunkten zu vorhandenen Klassen
  - anhand von Ähnlichkeiten bzw. übereinstimmender Merkmale
  - supervised learning

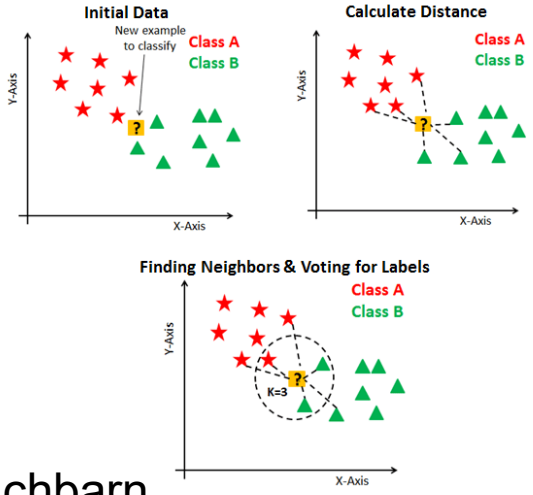


- Clusteranalyse:
  - Identifizierung von Ähnlichkeiten unter Datenstrukturen
  - Gruppierung dieser ähnlichen Datenpunkte
  - unsupervised learning

k-Nearest Neighbors (kNN) – Klassifikation

- Vorgehen:
  1. Berechnet die Distanz eines Punktes zu den Nachbarn
  2. Identifiziert die *k* Nachbarn mit den geringsten Abständen
  3. Entscheidet anhand der *k* Nachbarn über die Zugehörigkeit des betrachteten Punktes (welche Zugehörigkeit haben die Nachbarn?)

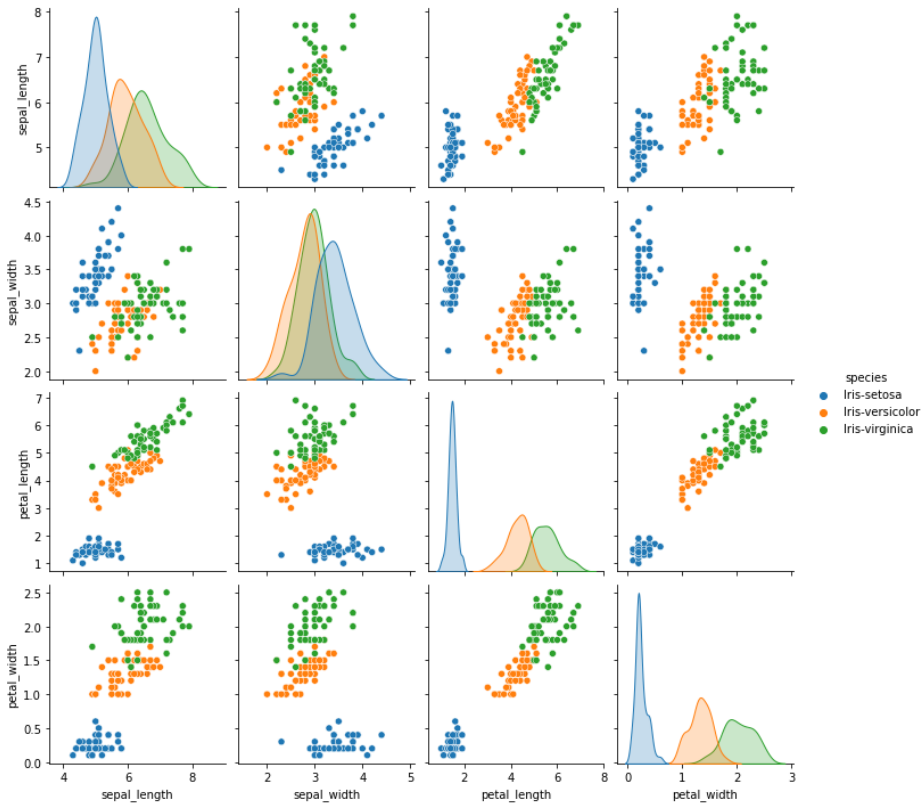




- Parameter (Auszug):
  - k: Anzahl zu untersuchender Nachbarn
  - Gewichtung: gleichwertig oder nach Distanz
  - Algorithmus: BallTree, KDTree, brute-force, auto

Exkurs: Eine erste Übersicht über den Datensatz

```python
# Bibliotheken laden
import seaborn as sns
import matplotlib.pyplot as plt

# Datensatz aus der Bibliothek seaborn laden
df = sns.load_dataset("iris")

# Eine erste Übersicht über den Datensatz
print(df.head())
print(df.describe())

# Den Pairplot erstellen und anzeigen
sns.pairplot(df, hue="species")
plt.show()
```



describe()

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

head()

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

k-Nearest Neighbors – Lernbeispiel [1/2]

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
import kNNplot

# Datensatz laden
iris = load_iris()

# Schlüssel anzeigen (Datenstruktur ist Dictionary)
print(iris.keys())
# Auswählen und Anzeigen der Features
X = iris.data
print(iris.feature_names)
print(iris.data.tolist())

# Auswählen und Anzeigen der Zielvariablen
y = iris.target
print(iris.target_names)
print(iris.target)

# Datensatz aufteilen in Trainings- und Testdaten
X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, random_state=142)
print("X_train:", X_train.shape, "X_test:", X_test.shape)
print("y_train:", y_train.shape, "y_test:", y_test.shape)
```

```
dict_keys(['data', 'target', 'frame', 'target_names',
           'DESCR', 'feature_names', 'filename', 'data_module'])
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
[[5.1, 3.5, 1.4, 0.2], [4.9, 3.0, 1.4, 0.2], [4.7, 3.2, 1.3, 0.2], [4. …
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 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, 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, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
X_train: (112, 4) X_test: (38, 4)
y_train: (112,) y_test: (38,)
```

PLRI PETER L. REICHERTZ INSTITUT FÜR MEDIZINISCHE INFORMATIK

k-Nearest Neighbors – Lernbeispiel [1/2]

```
28  # Den kNN-Klassifikator aussuchen und anwenden
29  kNN = KNeighborsClassifier(n_neighbors=5)
30  kNN.fit(X_train, y_train)
31
32  # Ausgabe der zu den einzelnen Datenpunkten zugeordneten Klassen
33  y_pred = kNN.predict(X_test)
34  print(y_pred)
35  # direkter Vergleich mit den tatsächlichen Klassen
36  print(y_test)
37
38  # Beispiel: Ausgabe der Klassenzugehörigkeit eines Datenpunktes
39  print(iris.target_names[y_pred[14]])
40  # direkter Vergleich mit der tatsächlichen Klasse
41  print(iris.target_names[y_test[14]])
42
43  # Beispiel: Ausgabe der Wahrscheinlichkeiten der Klassenzugehörigkeit
44  # hier: 60% virginica, 40% versicolor
45  print(kNN.predict_proba(X_test)[14])
46
47  # Erzeugen und Anzeigen der Konfusionsmatrix
48  print(confusion_matrix(y_test, y_pred))
49
50  # Die Genauigkeit ermitteln und ausgeben
51  accuracy = accuracy_score(y_test, y_pred)*100
52  print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')
```

```
array([0, 1, 1, 2, 1, 1, 0, 0, 2, 1, 1, 1, 2, 0, 2, 0, 2, 1, 1, 2, 2, 1,
       0, 1, 2, 2, 2, 2, 0, 1, 2, 1, 2, 2, 1, 2, 1, 2])

array([0, 1, 1, 2, 1, 1, 0, 0, 2, 1, 1, 1, 2, 0, 1, 0, 2, 1, 2, 1, 2, 1,
       0, 1, 2, 2, 2, 2, 0, 1, 2, 1, 2, 1, 1, 2, 1, 2])
```

```
'virginica'
'versicolor'
array([0. , 0.4, 0.6])
[[ 7  0  0]
 [ 0 14  3]
 [ 0  1 13]]
'Accuracy of our model is equal 89.47 %.'
```

Alle *setosa* wurden korrekt klassifiziert.
3 *versicolor* wurden als *virginica* klassifiziert.
1 *virginica* wurde als *versicolor* klassifiziert.

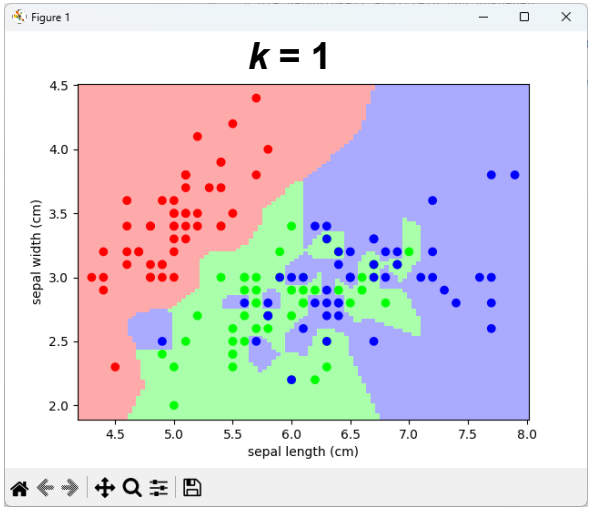
k-Nearest Neighbors – Abhängigkeit von *k* [1/2]

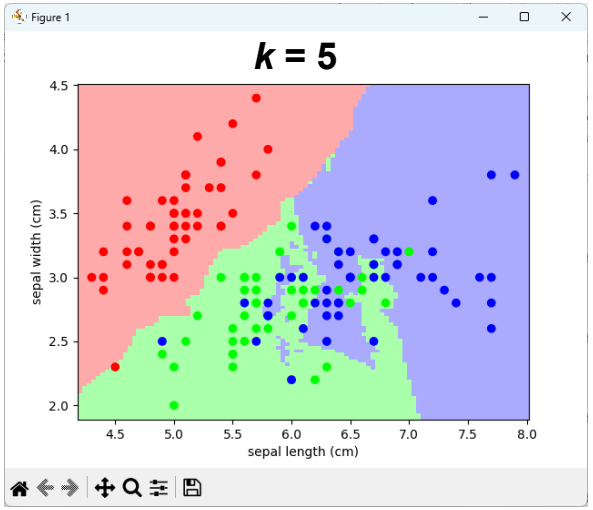
```python
# Die Klassifikation über 2 Parameter (sepal_length und sepal_width)
# jeweils mit k = 1, 5, 10, 15
X = iris.data[:, :2]

kNN = KNeighborsClassifier(n_neighbors=1)
kNN.fit(X, y)
kNNplot.plot_iris_knn(kNN, X, y)

kNN = KNeighborsClassifier(n_neighbors=5)
kNN.fit(X, y)
kNNplot.plot_iris_knn(kNN, X, y)

kNN = KNeighborsClassifier(n_neighbors=10)
kNN.fit(X, y)
kNNplot.plot_iris_knn(kNN, X, y)

kNN = KNeighborsClassifier(n_neighbors=15)
kNN.fit(X, y)
kNNplot.plot_iris_knn(kNN, X, y)
```

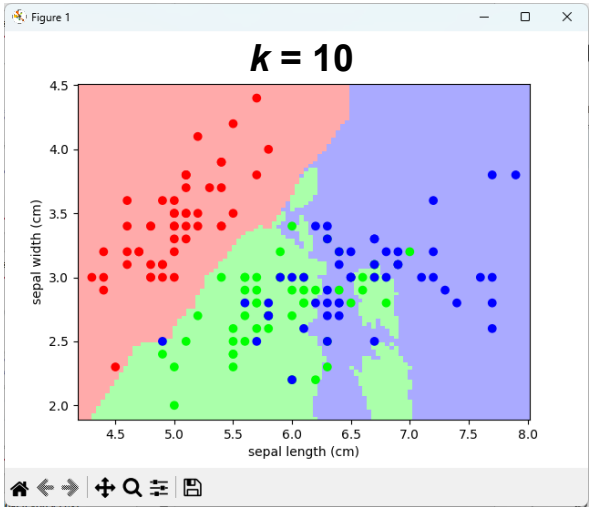
Inhalt der Datei ***kNNplot.py***

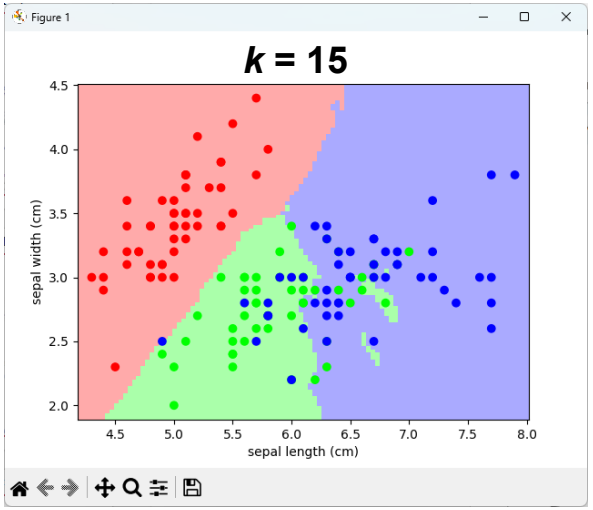
```python
import numpy as np
import pylab as pl
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])


4 usages
def plot_iris_knn(kNN, X, y):
    x_min, x_max = X[:, 0].min() - .1, X[:, 0].max() + .1
    y_min, y_max = X[:, 1].min() - .1, X[:, 1].max() + .1
    xx, yy = np.meshgrid( *xi: np.linspace(x_min, x_max, num: 100),
                          np.linspace(y_min, y_max, num: 100))
    Z = kNN.predict(np.c_[xx.ravel(), yy.ravel()])

    # Visualisierung der Klassifikation (Klassengrenzen)
    Z = Z.reshape(xx.shape)
    pl.figure()
    pl.pcolormesh( *args: xx, yy, Z, cmap=cmap_light)

    # Einzeichnen der einzelnen Datenpunkte (tatsächliche Klassen)
    pl.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold)
    pl.xlabel('sepal length (cm)')
    pl.ylabel('sepal width (cm)')
    pl.axis('tight')

    plt.show()
```

k-Nearest Neighbors – Abhängigkeit von *k* [2/2]



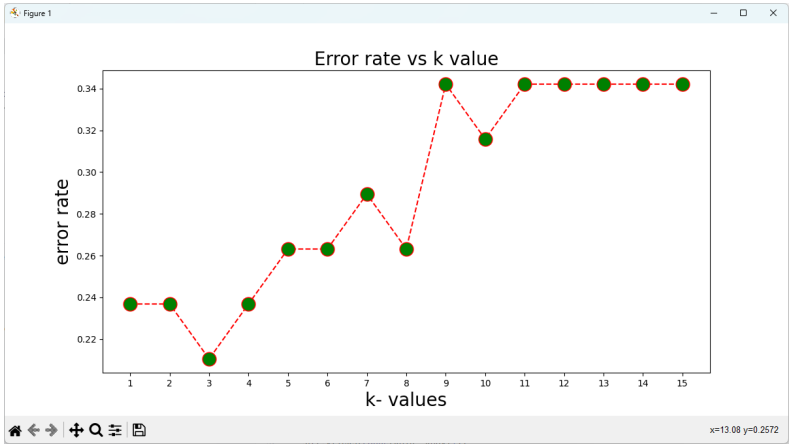






k-Nearest Neighbors – Das optimale *k*

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

iris = load_iris()

# zur Veranschaulichung wieder nur 2 Features (sepal_length und sepal_width)
X = iris.data[:, :2]
y = iris.target

X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, random_state=142)

# Das optimale k ermitteln
error_rate = []
kMin, kMax = 1, 15

for i in range(kMin, kMax+1):
    kNN = KNeighborsClassifier(n_neighbors=i)
    kNN.fit(X_train, y_train)
    y_pred = kNN.predict(X_test)
    error_rate.append(np.mean(y_pred != y_test))

# Die Ergebnisse ausgeben
plt.figure(figsize=(12, 6))
plt.plot( *args: range(kMin, kMax+1), error_rate, marker="o", markerfacecolor="green",
         linestyle="dashed", color="red", markersize=15)
plt.title( label: "Error rate vs k value", fontsize=20)
plt.xlabel( xlabel: "k- values", fontsize=20)
plt.ylabel( ylabel: "error rate", fontsize=20)
plt.xticks(range(kMin, kMax+1))
plt.show()
```



k-Nearest Neighbors – Fazit

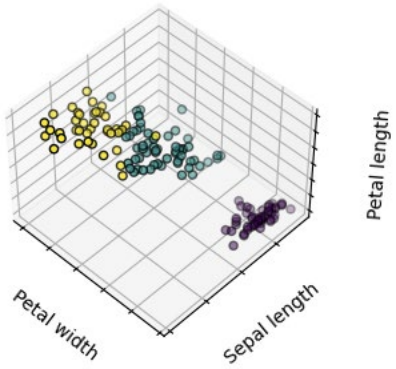
- Auch als Regression möglich

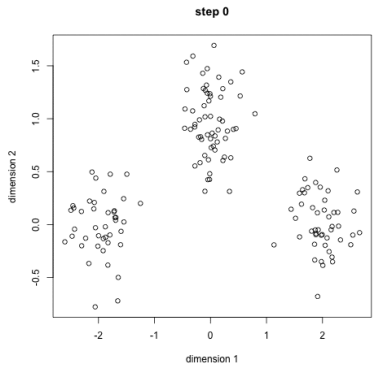
- Vorteile:
  - Einfache Anwendung (kann einfach erlernt/nachvollzogen werden)
  - Einfache Anpassung an neue Daten (nach demselben Muster)
  - Wenige notwendige Parameter (alles außer Daten und k ist optional)

- Nachteile:
  - Vergleichsweise langsam und ressourcenhungrig
  - Für hochdimensionale Daten nur schlecht geeignet
  - Anfälligkeit (niedriges *k* führt zu Overfitting,
    hohes *k* führt zu Glättung / Unteranpassung)

k-Means / k-Median – Clusteranalyse

- Vorgehen:
  1. Bestimme die Anzahl *k* der gewünschten Cluster
  2. Bestimme die Mittelpunkte / Mediane der einzelnen Cluster (random)
  3. Ordne jeden Datenpunkt dem nächstgelegenen Mittelpunkt / Median zu
  4. Berechne die Mittelpunkte / Mediane der einzelnen Cluster neu
  5. Wiederhole 3.-4., solange die Mittelpunkte / Mediane sich verändern





- Parameter (Auszug):
  - k: Anzahl erwünschter Cluster
  - n_init: Anzahl der Initialisierungen
  - max_iter: Anzahl der Iterationen pro Initialisierung

k-Means – Lernbeispiel [1/2]

```python
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import kMplot

# Datensatz laden
iris = load_iris()

# Auswählen und Anzeigen der Features
X = iris.data
print(iris.feature_names)
print(iris.data.tolist())

# Den kNN-Klassifikator aussuchen und anwenden
kM = KMeans(n_clusters=3)
kM.fit(X)

# Die Cluster ausgeben
print(kM.labels_) # kM.predict(X)

# die Clusterzentren ausgeben
print(kM.cluster_centers_)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

```
[[5.1, 3.5, 1.4, 0.2], [4.9, 3.0, 1.4, 0.2], [4.7, 3.2, 1.3, 0.2], [4. …
```

```
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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0,
       0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2])
```

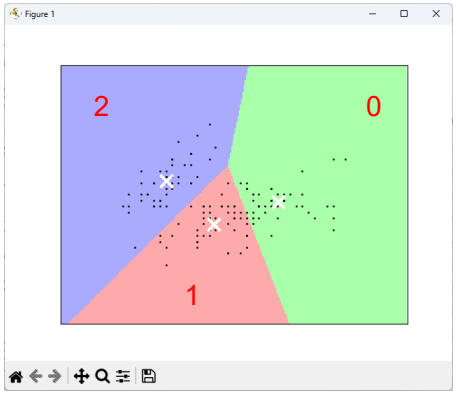
```
array([[6.85      , 3.07368421, 5.74210526, 2.07105263],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [5.9016129 , 2.7483871 , 4.39354839, 1.43387097]])
```

k-Means – Lernbeispiel [2/2]

```python
# Das Clustering über 2 Parameter (sepal_length und sepal_width)
# jeweils mit k = 1, 5, 10, 15
X = iris.data[:, :2]

kM = KMeans(n_clusters=3)
kM.fit(X)

print(kM.labels_)
print(kM.cluster_centers_)

kMplot.plot_iris_kM(kM, X)
```

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1])
```

```
array([[6.81276596, 3.07446809],
       [5.77358491, 2.69245283],
       [5.006     , 3.428     ]])
```



Inhalt der Datei ***kMplot.py***

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])


1 usage
def plot_iris_kM(kM, X):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid( *xi: np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))
    Z = kM.predict(np.c_[xx.ravel(), yy.ravel()])

    # Visualisierung des Clustering (Clustergrenzen)
    Z = Z.reshape(xx.shape)
    plt.figure(1)
    plt.clf()
    plt.imshow(Z, interpolation="nearest", extent=(xx.min(), xx.max(), yy.min(), yy.max()),
               cmap=cmap_light, aspect="auto", origin="lower")
    plt.plot( *args: X[:, 0], X[:, 1], "k.", markersize=2)

    # Visualisierung der Clusterzentren
    centroids = kM.cluster_centers_
    plt.scatter(centroids[:, 0], centroids[:, 1], marker="x", s=169, linewidths=3,
                color="w", zorder=10)
    plt.xlim( *args: x_min, x_max)
    plt.ylim( *args: y_min, y_max)
    plt.xticks(())
    plt.yticks(())
    plt.show()
```

kMeans – Das optimale *k*

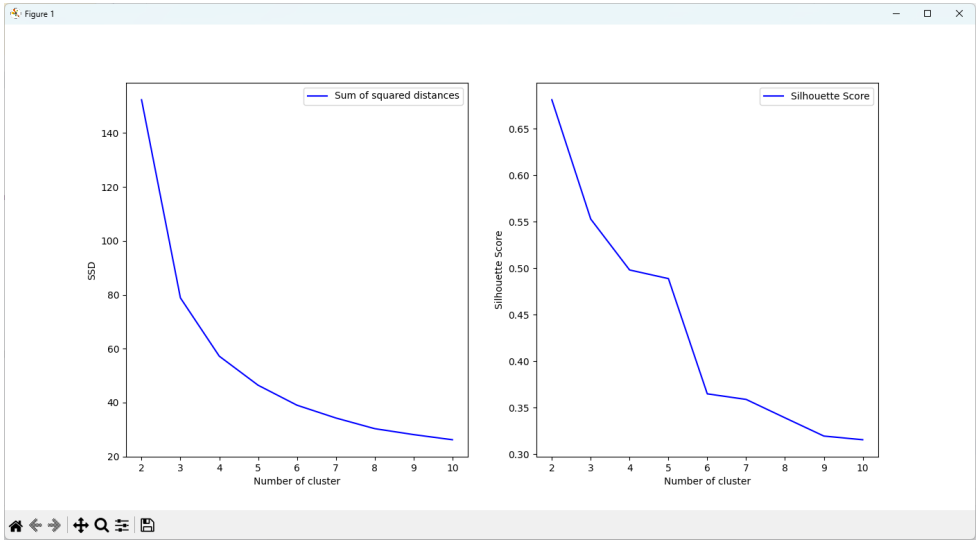
- Mittels Ellenbogenkurve
  - hier wird ermittelt, ab wann sich ein höherer Aufwand (ein höheres k) nicht mehr rechnet bzw. das Clustering nicht mehr verbessert
  - es wird pro k die Summe der Abstände der einzelnen Datenpunkte zu Ihrem nächsten Clusterzentrum zum Quadrat berechnet
  - dadurch kann abgelesen werden, welche Vergrößerung des k welche Verbesserung dieses Maßes hervorruft

- Mittels Silhuettenkoeffizient
  - hier wird die Nähe der einzelnen Datenpunkte zum eigenen Cluster im Gegensatz zu der Nähe der einzelnen Datenpunkte zum jeweiligen nächsten Nachbarcluster ermittelt
  - ein Koeffizient nahe 1 besagt, dass die Zuteilungen vermutlich gut sind
  - ein Koeffizient nahe -1 besagt das Gegenteil

Die Entscheidung liegt dann beim Betrachter!

kMeans – Das optimale *k*

```python
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data

# Das optimale k ermitteln
elbow, ss = [], []
kMin, kMax = 2, 10

for i in range(kMin, kMax+1):
    kM = KMeans(n_clusters=i)
    y_pred = kM.fit_predict(X)
    silhouette_avg = silhouette_score(X, y_pred)
    ss.append(silhouette_avg)
    elbow.append(kM.inertia_)

# Die Ergebnisse ausgeben
fig = plt.figure(figsize=(14, 7))
fig.add_subplot(121)
plt.plot( *args: range(kMin, kMax+1), elbow, 'b-', label='Sum of squared distances')
plt.xlabel("Number of cluster")
plt.ylabel("SSD")
plt.legend()
fig.add_subplot(122)
plt.plot( *args: range(kMin, kMax+1), ss, 'b-', label='Silhouette Score')
plt.xlabel("Number of cluster")
plt.ylabel("Silhouette Score")
plt.legend()
plt.show()
```



Aufgrund der Ähnlichkeiten anhand der Merkmale wäre eine Einteilung in 2, 3 oder 4 Cluster denkbar.

Das Ergebnis ist hier nicht eindeutig und weicht von der in diesem Beispiel bekannten Anzahl an Arten (3) ab.
(d.h. in diesem Beispiel könnten auch andere logische Einteilungen vorgenommen werden als die 3 bekannten Arten)

k-Means / k-Median – Fazit

- k-Median ist weniger anfällig für Ausreißer

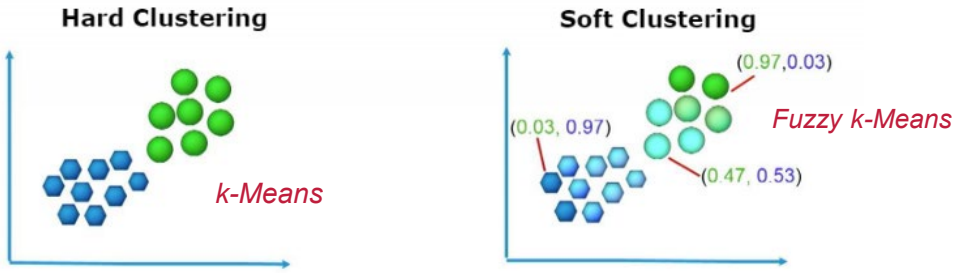
- Vorteile:
  - Relativ einfache Implementierung
  - Einfache Anpassung an neue Daten (nach demselben Muster)
  - Skalierbar auf große Datensätze

- Nachteile:
  - das optimale *k* muss zuerst gefunden werden
  - Probleme bei Clustern unterschiedlicher Größe, Form und Dichte
  - „Fluch der Dimensionalität“

Fuzzy k-Means – Clusteranalyse

- <u>Fuzzy Logik:</u>
  - Unschärfelogik
  - Unterscheidung nach Gewichtung statt eindeutiger Zuordnung

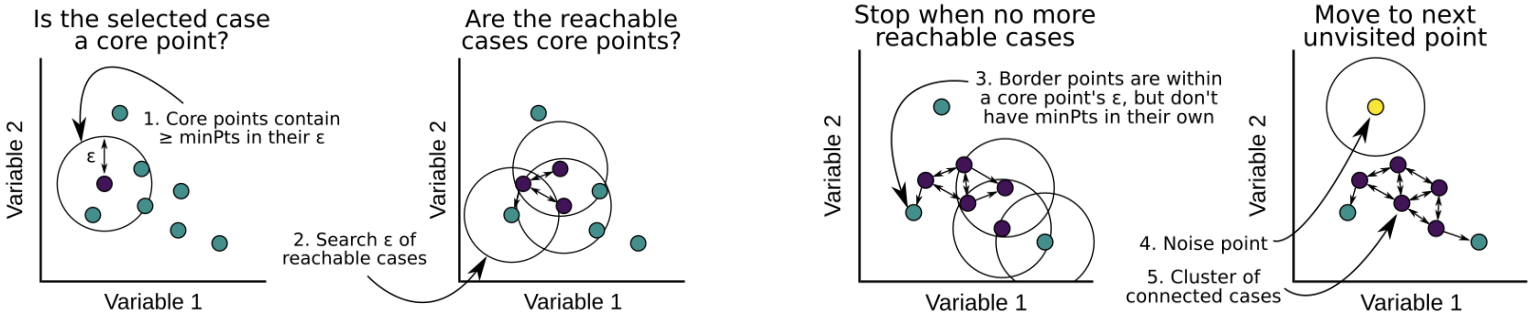
- Erweiterung des k-Means – Algorithmus
- Die einzelnen Datenpunkte werden nicht genau einem Cluster zugeordnet, sondern mit Gewichten je Cluster versehen
- Diese Gewichte geben an, wie stark die Zugehörigkeit des Datenpunktes zum jeweiligen Cluster ist
- Entscheidung aufgrund der einzelnen Gewichte



DBScan – Clusteranalyse (unsupervised)

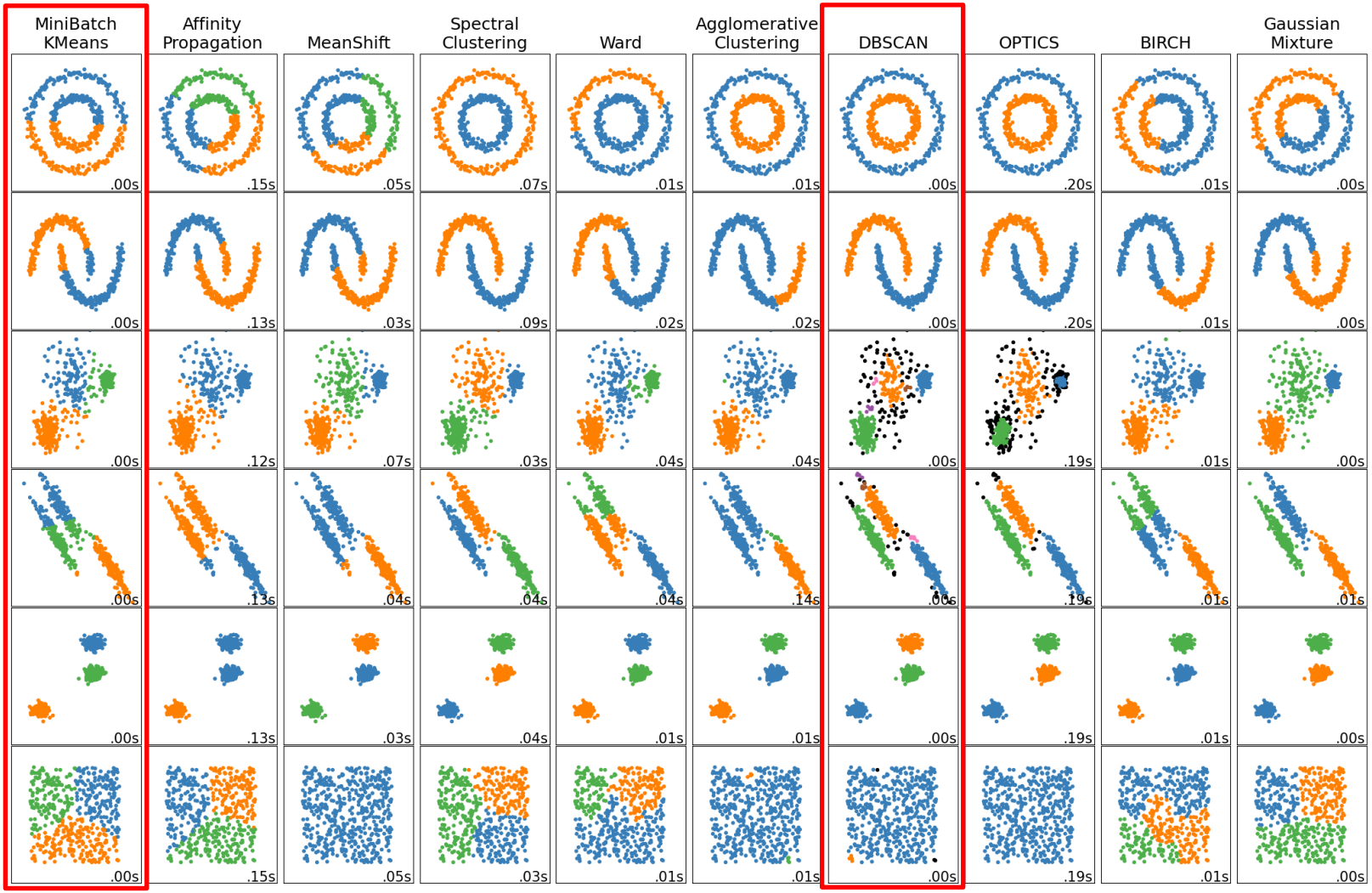
- Density-Based Spatial Clustering of Applications with Noise
- → “Dichtebasierte räumliche Clusteranalyse mit Rauschen“

- Zuordnung der einzelnen Datenpunkte nach Dichteverbundenheit
- Automatische Erkennung der Clusteranzahl
- Erkennung von Clustern beliebiger Form



- <u>Parameter:</u>
  - Epsilon: max. Nachbarschaftslänge eines Datenpunktes
  - minPts: min. Anzahl an Nachbarn

Vergleich der Cluster – Algorithmen (aus sklearn)



PETER L. REICHERTZ INSTITUT FÜR MEDIZINISCHE INFORMATIK

Literatur

- sklearn.neighbors.KNeighborsClassifier — scikit-learn 1.3.1 documentation
- sklearn.cluster.KMeans — scikit-learn 1.3.1 documentation

- Machine Learning 101 — Classification vs. Clustering | by Kevin C Lee | Medium
- K-Nearest neighbor clustering — Machine learning book (vatsalparsaniya.github.io)
- The 5 Clustering Algorithms Data Scientists Need to Know | by George Seif | Towards Data Science
- API Reference — scikit-learn 1.1.2 documentation
- Fuzzy C-Means Clustering —Is it Better than K-Means Clustering? | by Satyam Kumar | Towards Data Science
- DBSCAN Clustering in ML | Density based clustering – GeeksforGeeks
- Tutorial for DBSCAN Clustering in Python Sklearn - MLK - Machine Learning Knowledge
- 18 Clustering based on density: DBSCAN and OPTICS - Machine Learning with R, the tidyverse, and mlr (manning.com)
- Understanding DBSCAN Algorithm and Implementation from Scratch | by Andrewngai | Towards Data Science