Fachhochschule Graubünden
University of Applied Sciences

Methods & Algorithms

(formerly known as Introduction to Data Science)

FS 2023

Prof. Dr. rer. nat. habil. Ralf-Peter Mundani

DAViS

Eine Geschichte von Nachbarn und Mitten

- worum geht es…?
  - Klassifizierung von Daten
  - überwachtes Lernen vs. unüberwachtes Lernen

    $\rightarrow$ wir kennen die Antwort vs. wir wissen nichts über die Antwort

- Verfahren
  - *k*-Nächste-Nachbarn
  - *k*-Mitten (*k*-Mittelpunkte)

> *Data scientist (noun): Person who is better at statistics than any software engineer and better at software engineering than any statistician.*
>
> — Josh Wills

k-Nächste-Nachbarn

- Ausgangssituation
  - Punktewolke von bereits klassifizierten Objekten (z.B. Krebsrisiko bei Patienten)
  - neue, unklassifizierte Patienten sollen aufgenommen und klassifiziert werden
  - Frage: lineare Regression ja oder nein…?

- LR grundsätzlich möglich, ABER
  - keine kontinuierliche, sondern kategoriale Variable
  - Modellierung als verschiedene Bereiche (z.B. 'gering', 'mittel', 'hoch', 'sehr hoch', …)
  - nicht für alle Probleme umsetzbar (z.B. 'vermutlich Demokrat', 'vermutlich Republikaner', …)
  - anderes Konzept notwendig $\rightarrow$ *k-NN*

k-Nächste-Nachbarn

- Idee *k-NN*
  1) betrachte *ähnliche* Objekte (im Hinblick auf ihre Attribute)
  2) entscheide bzgl. deren Klassifizierung (Mehrheitsentscheid)
  3) bei Gleichstand entscheide zufällig (aus den Mehrheiten)

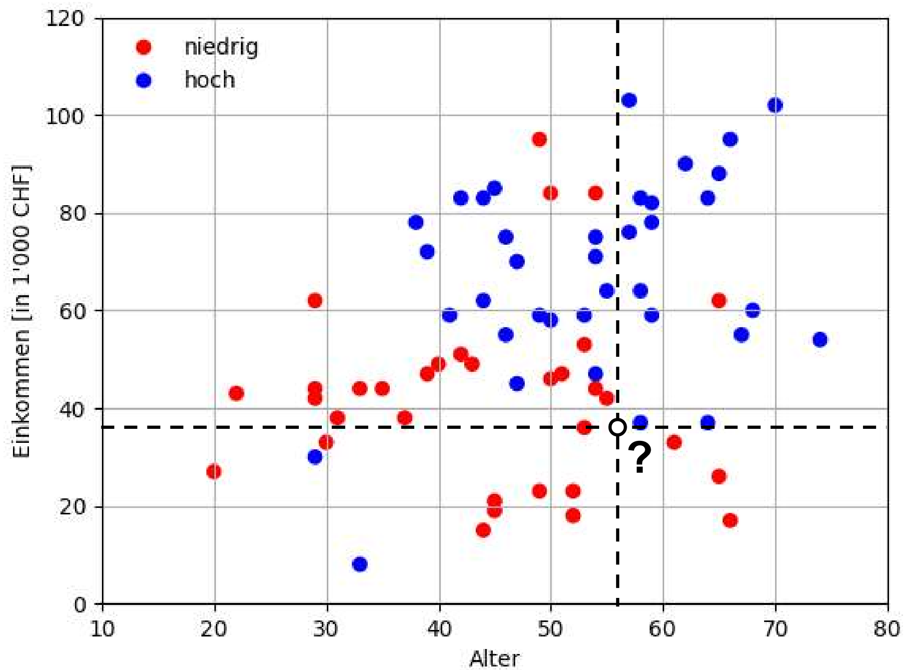
- Beispiel: Filmbeurteilung (★★★★★)
  - neuer Film 'Episode VII: The Data Scientist Awakens'
  - Attribute: Länge, Genre, Darsteller, Budget, …
  - suche *ähnliche* Filme und betrachte deren Klassifizierung (bspw. 8 ähnliche Filme mit ★★★★★ und 5 ähnliche Filme mit ★★★★★)
  - Ergebnis: ★★★★★ (und dass, ohne den Film jemals gesehen zu haben ☺)

k-Nächste-Nachbarn

- Überlegungen
  - wie lässt sich Ähnlichkeit definieren…? → nächste Nachbarn (Metrik)
  - wie viele nächste Nachbarn sollen berücksichtigt werden → *k* Stück

- Beispiel: Bonität (Datei 'credit.dat')
  - Alter, Einkommen, Bonität
  - Darstellung als Streudiagramm
  - 70 beliebige Personen

- Frage: neue Person (57) mit 37'000 CHF Einkommen → niedrig/hoch…?



k-Nächste-Nachbarn

- Daten einlesen und plotten

```
from matplotlib.colors import ListedColormap

data = np.loadtxt('credit.dat', delimiter=',')
col = ListedColormap(['red', 'blue'])
sp = plt.scatter(data[:,0], data[:,1], c=data[:,2], cmap=col)
lab = ['niedrig', 'hoch']
plt.legend(handles=sp.legend_elements()[0], labels=lab)
plt.show()
```

Farbsequenz (hier 0 oder 1)

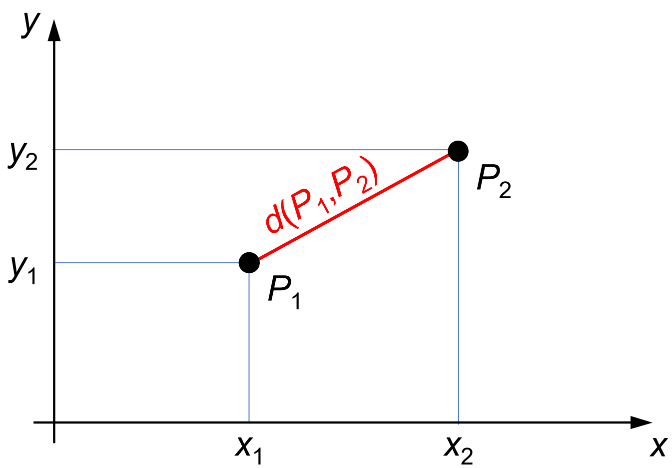
k-Nächste-Nachbarn

- Umsetzung / Einzelschritte
    1) Ähnlichkeits- oder Distanzmetrik
    2) Klassifikationsalgorithmus
    3) Festlegung Bewertungsmass
    4) Aufteilung der klassifizierten Daten in Trainings- und Testdaten
    5) mehrere Durchläufe *k-NN* für unterschiedliche Werte von *k*
    6) wähle *k* bzgl. bester erreichter Bewertung
    7) Klassifikation neuer Daten

- Prinzip / Theorie soweit klar, ab zur Mathematik… ☺

k-Nächste-Nachbarn

- Ähnlichkeits- / Distanzmetrik (1)
  - Euklidischer Abstand zweier Punkte P_1, P_2 in der Ebene



$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Vorsicht: Skalierungseffekte
  - Alter / Einkommen: (25, 54'000) und (35, 76'000) $\leftrightarrow$ (25, 54) und (35, 76)

k-Nächste-Nachbarn

- Ähnlichkeits- / Distanzmetrik (2)
  - Jaccard'scher Abstand / Ähnlichkeit
    - Distanz zwischen einer Menge von Objekten
    - Urs' Freunde $A = \{$Rita, Sarah, Patrick, $\ldots\}$, Lauras Freunde $B = \{$Patrick, Rita, Lisa, $\ldots\}$

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

  - Hamming Abstand
    - Anzahl der Stellen, an denen sich zwei gleich lange Sequenzen unterscheiden
    - A = SCHWEIZ, B = ITALIEN, C = BELGIEN

$$H(A,B) = 7, \qquad H(B,C) = 4$$

Vorlesung schauen

k-Nächste-Nachbarn

- Ähnlichkeits- / Distanzmetrik (3)
  - Cosinus Ähnlichkeit
    - beschreibt Ähnlichkeit zweier Vektoren $\vec{x}, \vec{y}$
    - Wertebereich: -1 (exakt entgegengesetzt) bis 1 (exakt gleich)

$$\cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{||\vec{x}|| \; ||\vec{y}||}$$

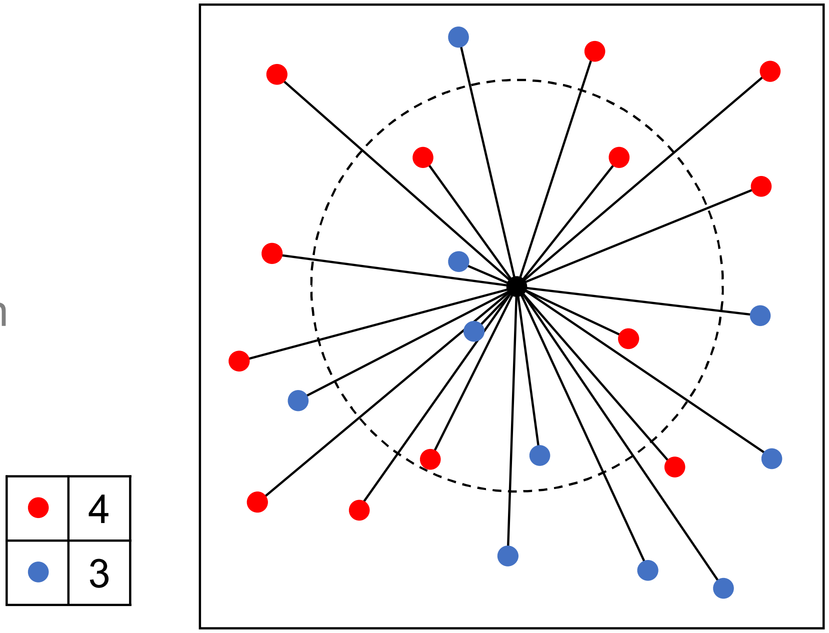
  - Manhattan Abstand
    - Abstand zweier *k*-dimensionaler Vektoren $\vec{x}, \vec{y}$
    - Idee: Taxi fährt durch die Strassen von Manhattan (Gitter)

$$d(\vec{x}, \vec{y}) = \sum_{i=1}^{k} |x_i - y_i|$$

Quelle: forgotten-ny.com

k-Nächste-Nachbarn

- Klassifizierung (Mehrheitsentscheid)
  - neues Element einfügen
  - Berechnung aller Distanzen (~O(*n*))
  - Auffinden der *k* (= 7) nächsten Nachbarn
  - Bestimmung der zugehörigen Kategorien
  - Mehrheitsentscheid → Klassifikation (●)



k-Nächste-Nachbarn

- Bewertungsmass
  - Treffsicherheit
    - berechnet Anteil resp. Anzahl korrekter Vorhersagen $\hat{y}_i$
    - Indikatorfunktion 1($\cdot$) $\rightarrow \{0, 1\}$ mit 1 falls $\hat{y}_i = y_i$ und 0 falls $\hat{y}_i \neq y_i$

$$accuracy(y, \hat{y}) = \frac{1}{n} \sum_i 1(\hat{y}_i = y_i)$$

  - Hamming Verlust (Fehlinterpretationsrate)
    - berechnet durchschnittliche(n) Distanz / Verlust zwischen zwei Mengen

$$L_{Hamming}(y, \hat{y}) = \frac{1}{n} \sum_i 1(\hat{y}_i \neq y_i) \equiv 1 - accuracy(y, \hat{y})$$

k-Nächste-Nachbarn

- Modell aufsetzen und trainieren (Kreuzvalidierung)

```
from sklearn.neighbors import KNeighborsClassifier as knn
from sklearn.model_selection import train_test_split as tts
from sklearn.metrics import accuracy_score as score
x_train, x_test, y_train, y_test = tts(data[:,:2], data[:,2],
   test_size=0.3)
```
→ 70% Trainingsdaten, 30% Testdaten

```
model = knn(n_neighbors=k)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
score(y_test, y_pred)
```

k-Nächste-Nachbarn

- Bestimmung des optimalen *k*-Werts aus [1, *n*]

```
for k in range(1,16):
  model = knn(n_neighbors=k)
  model.fit(x_train, y_train)
  y_pred = model.predict(x_test)
  print(k, ':', score(y_test, y_pred))
```

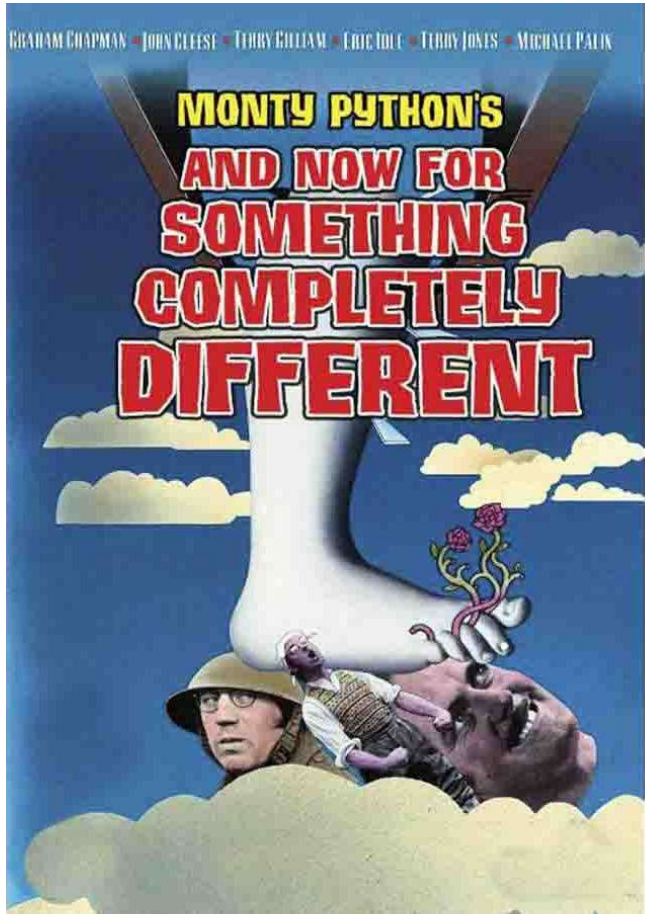
- Frage: welche Werte für *k* sinnvoll…?

- und welche Bonität hat nun unsere Person (57) mit 37'000 CHF Einkommen…?

`model.predict([[57., 37.]])` → niedrig

k-Nächste-Nachbarn

- noch ein paar Gedanken zu **`KNeighborsClassifier`**
  - Parameter: **`algorithm`**
    - ball_tree
    - kd_tree
    - brute (force)
  - Parameter: **`metric`**
    - euclidean
    - minkowski (default; $p = 2$): $d(\vec{x}, \vec{y}) = \left(\sum_i |x_i - y_i|^p\right)^{\frac{1}{p}}$
    - manhattan
    - …



Quelle: montypython.com

k-Mitten

- was bisher geschah…
  - überwachtes Lernen (*k-NN*)
  - 'korrekte' Antwort (d.h. Klassifizierung) a-priori bekannt
  - Modell damit bestmöglich trainiert (hohe Treffsicherheit ./. geringer Verlust)

- was ist jetzt anders…?
  - Ausgangslage: unklassifizierte Daten
  - 'korrekte' Antwort resp. irgendeine Antwort nicht bekannt
  - überwachtes Lernen nicht anwendbar $\rightarrow$ unüberwachtes Lernen
  - Algorithmus der Stunde: *k*-Mitten (engl. *k-means*)

k-Mitten

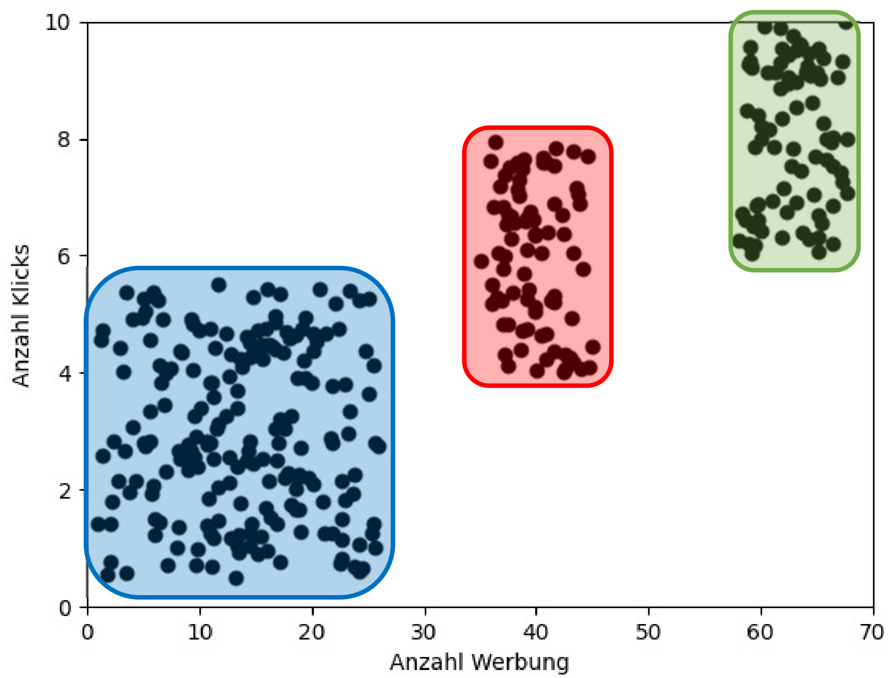
- Idee
  - *Klassifikation* der Daten durch Clusterbildung (Gruppen)
  - Cluster sollen Elemente mit ähnlichen Attributen enthalten

- Gedankenexperiment: manuelle Eingruppierung
  - Daten enthalten Alter, Geschlecht, Einkommen, Kanton, Haushaltsgrösse
  - Festlegung von Bereichen / Gruppen (z.B. Alter 20–24, 25–29, …) `manuelle Gruppierung`
  - Ergebnis: Alter (10), Geschlecht (3), Einkommen (20), Kanton (26), Haushaltsgrösse (3)
  - daraus folgt: 5-dimensionaler Raum mit $10 \times 3 \times 20 \times 26 \times 3 = 46'800$ Einträgen

  ➔ Ausweg: Gruppierung der Gruppen (der Gruppen (der Gruppen (…)))

k-Mitten

- Beispiel: Klick-Rate (Datei 'clicks.dat')
  - der Einfachheit halber wieder in 2D (d.h. Daten mit nur zwei Attributen)
  - Ausgangslage: Nutzerverhalten bzgl. angezeigter Werbung (x) und Anzahl Klicks (y)
  - Darstellung als Streudiagramm
  - 360 beliebige Nutzer:innen



offenkundige (!) Cluster

k-Mitten

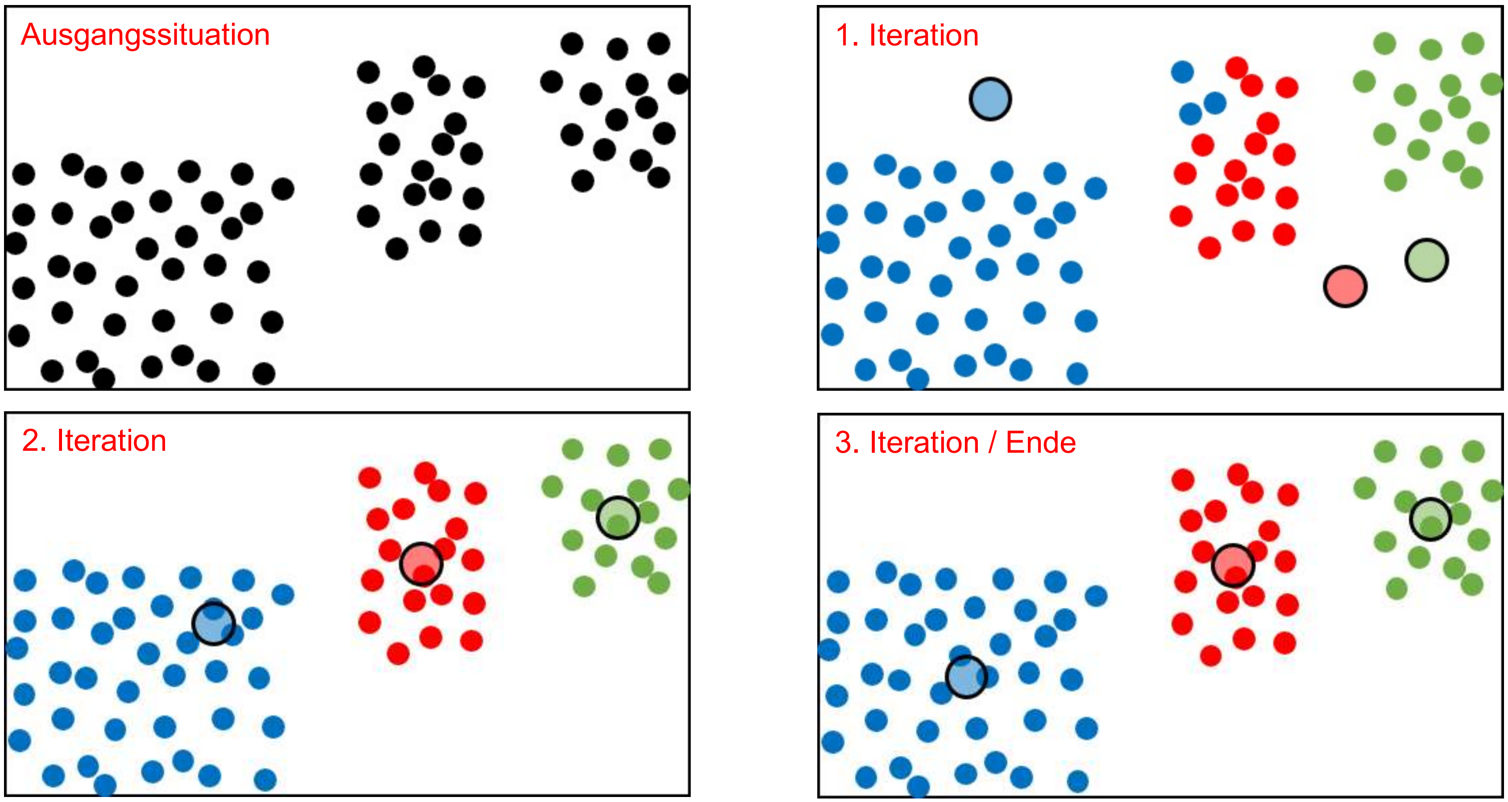
- Umsetzung / Einzelschritte

  1) Überlegung, wie viele Cluster könnte es geben $\rightarrow$ *k* Stück (mit $1 \leq k \leq n$)
  2) Auswahl *k* zufälliger Schwer- / Mittelpunkte (möglichst gut verteilt)
  3) Zuweisung jedes Datenpunktes zum nächstgelegenen Mittelpunkt
  4) Verschiebung der Mittelpunkte in die Mitte aller zugewiesenen Datenpunkte
  5) Wiederholung der Schritte 3 und 4 bis keine / wenig Änderung erfolgt

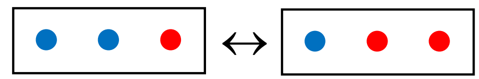
- was wird benötigt…?
  - Zufallsgenerator
  - Distanzmetrik
  - ein bisschen Mathematik ☺

k-Mitten

- nochmal Beispiel: Klick-Rate (vereinfacht; $k = 3$)



k-Mitten

- noch ein paar Worte zum Algorithmus (bevor es losgeht)
  - Wahl von *k* ist mehr Kunst als Wissenschaft ☺
  - Konvergenz ist nicht garantiert
    - Algorithmus oszilliert zwischen zwei Lösungen 
    - d.h. es gibt keine eindeutige Lösung
  - Interpretation der Lösung oftmals fragwürdig (bzw. nicht nützlich)
  - ABER: im Vergleich zu anderen Cluster-Algorithmen ist *k*-Mitten sehr schnell

- unnützes Wissen
  - 1957: erster Verweis auf den Algorithmus; Arbeit von H. Steinhaus & S. Lloyd
  - 1967: erste Verwendung des Begriffs *k-means*; 1982: erstmalige Publikation durch Bell Labs

k-Mitten

- Modell aufsetzen und Daten **`data`** klassifizieren

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=k, max_iter=value)
y_pred = model.fit_predict(data)
```

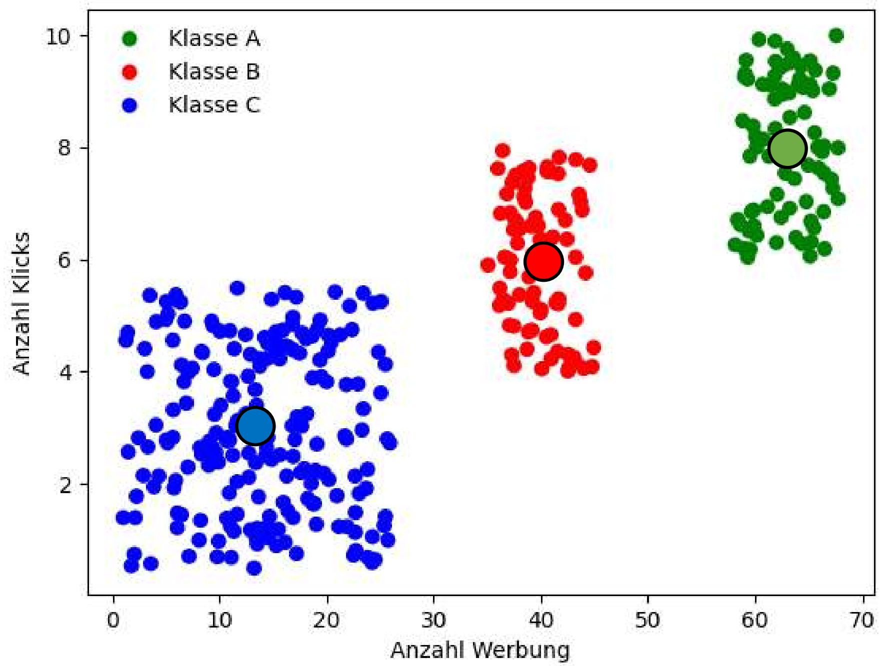
→ max. Anzahl Iteration bevor Abbruch

- Daten als Streudiagramm plotten

```
col = ListedColormap(['red', 'blue', 'green'])
sp = plt.scatter(data[:,0], data[:,1], c=y_pred, cmap=col)
lab = (['C1', 'C2', 'C3']
plt.legend(handles=sp.legend_elements()[0], labels=lab)
plt.show()
```

k-Mitten

- Überraschung…oder doch nicht…?



- und was, wenn wir den *k*-Wert nicht a-priori kennen…?

k-Mitten

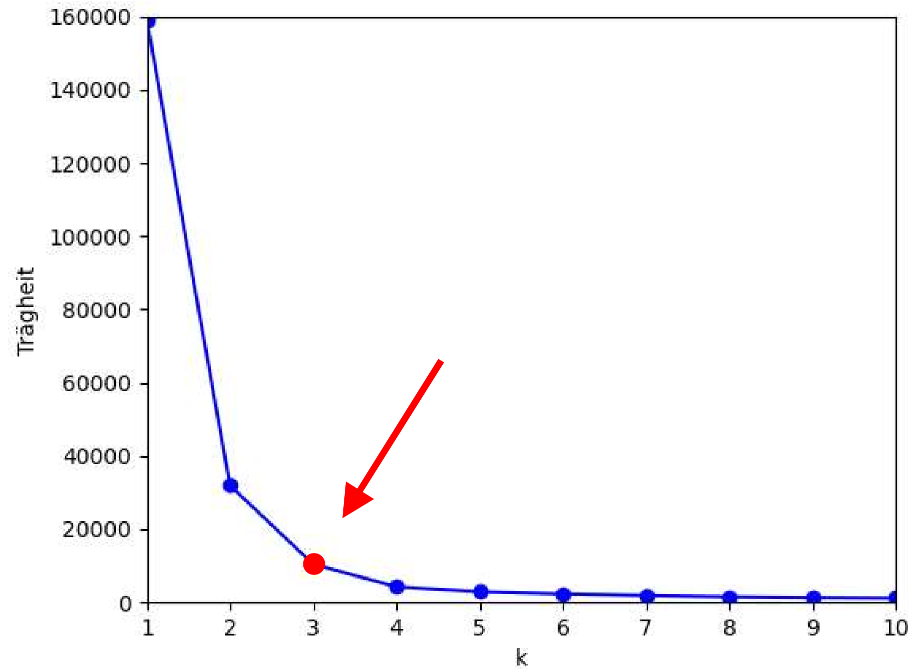
- Ellenbogen-Methode
  - *k*-Mitten versucht, die Trägheit (*inertia*) bzgl. der Mittelpunkte μ_j zu minimieren

$$I = \sum_i \min_{\mu_j \in C}(||x_i - \mu_j||^2)$$

  - Trägheit des Modells: **`model.inertia_`**
  - guter *k*-Wert am Knick des Ellenbogens

Berechnung

```
inert = []
for k in range(1,11):
  model = Kmeans(n_clusters=k)
  model.fit(data)
  inert.append(model.inertia_)
```



k-Mitten

- wir brauchen bessere Daten…
  - Paket **`sklearn`** bietet hierzu eine Vielfalt an Möglichkeiten
    - Beispiel 1: Kleckse (*blobs*)

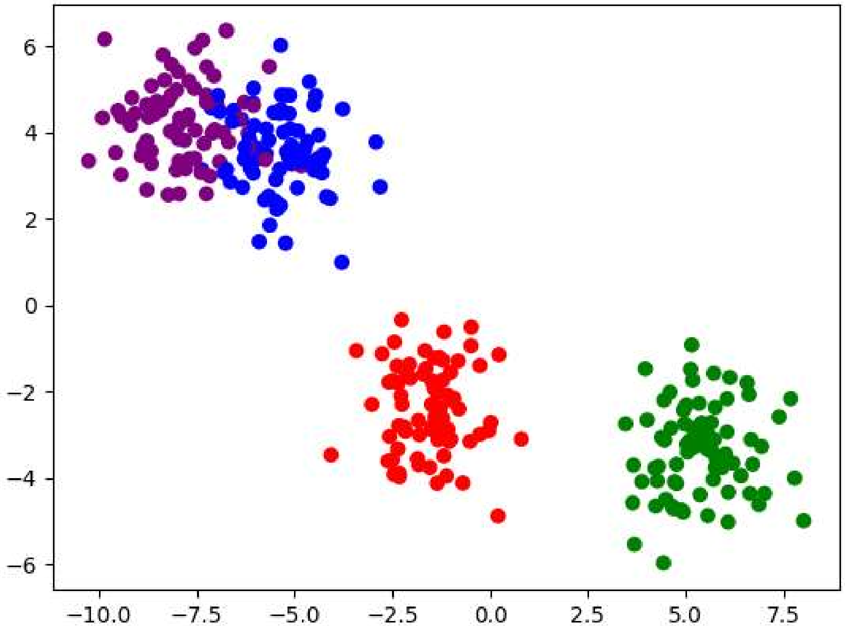
```
from sklearn.datasets import make_blobs
x, y_true = make_blobs(n_samples=value, centers=value)
```

- Beispiel 2: Monde

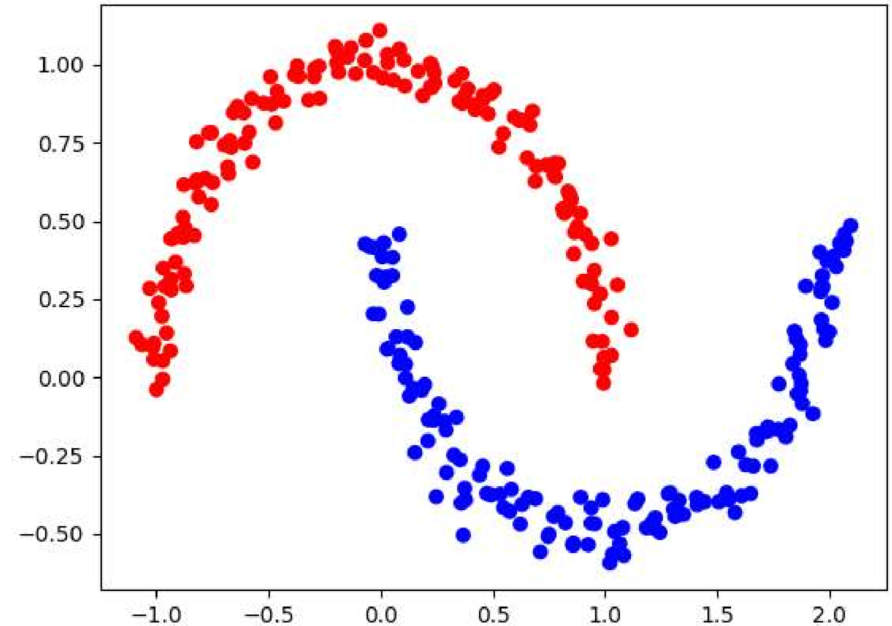
```
from sklearn.datasets import make_moons
x, y_true = make_moons(n_samples=value, noise=value)
```

k-Mitten

- wir brauchen bessere Daten… et voilà!



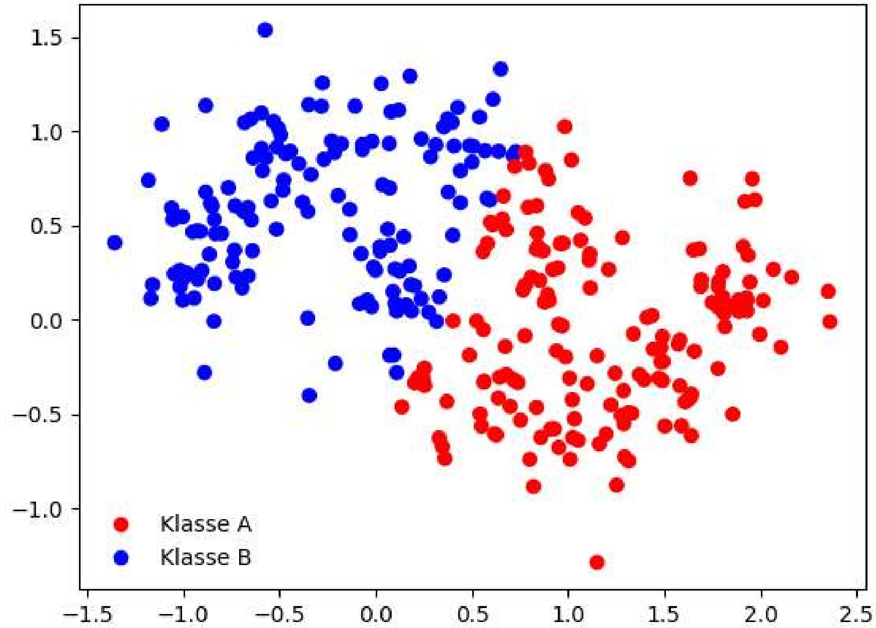
blobs: samples = 300; centers = 4



moons: samples = 200; noise = 0.05

Praktische Übung

- Aufgabe
  - Datensatz erzeugen (**`blobs`** oder **`moons`**)
  - mittels Ellenbogen-Methode guten *k*-Wert bestimmen
  - Ergebnisse plotten
  - Bsp: samples = 300; noise = 0.2 (*moons*)



- Zusatzfrage
  - Klassifizierung (y_true) bekannt
  - lässt sich Treffsicherheit bestimmen…?
  - Problem: unterschiedliche Bezeichner

Ein kleiner Blick über den Tellerrand…

#political_incorrect

- die Lena-Story (nicht Meyer-Landrut oder Gercke)
  - DAS meistverwendete Testbild in der Bildverarbeitung
  - A. Sawchuk benötigt Testbild für Kompressionsalgorithmus
  - vorhandene Testbilder (meistens zu matt) aus den 60ern
  - Kollege mit Ausgabe PB 11/1972 (Lena Söderberg)
  - nach Urheberrechtsstreitigkeiten willigt PB ein
  - 1997 wurde Lena zum 50. Geburtstag der SIST-Konferenz eingeladen
  - Lena-Testbild war lange Bestandteil des Pakets `scipy` (wurde inzwischen entfernt)

Quelle: lenna.org

- unnützes Wissen: lenPEG-Algorithmus ☺
  - falls Input = 'lena.png' → schreibe '0', andernfalls komprimiere Bild

Ein kleiner Blick über den Tellerrand…

- Bild einlesen (Paket **`matplotlib`**) und anzeigen

```
img = plt.imread('papa_color.png')
plt.imshow(img)
plt.show()
```

© R.-P. Mundani

'Lora' statt 'Lena' ☺

Ein kleiner Blick über den Tellerrand…

- Experiment: Farbraumreduzierung
  - Echtfarbenbild (16.8 Millionen Farben; RGB) auf 16 Farben reduzieren

```
img = plt.imread('papa_color.png')
data = np.reshape(img, (512*512, 3))

from sklearn.cluster import KMeans
model = KMeans(16)
model.fit(data)
data_reduced = model.cluster_centers_[model.predict(data)]
img16 = np.reshape(data_reduced, (512, 512, 3))
plt.imshow(img16)
plt.show()
```

Ein kleiner Blick über den Tellerrand…

- Experiment: Farbraumreduzierung
  - Echtfarbenbild (16.8 Millionen Farben; RGB) auf 16 Farben reduzieren

Original (16.8 Millionen Farben)

16 Farben

Fragen…?