# k-Nearest Neighbors Algorithmus

Prof. Dr. Jörg Frochte

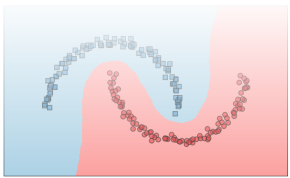
Maschinelles Lernen

$$\|x\| = 0 \Rightarrow x = \mathbf{0}$$
$$\|\alpha \cdot x\| = |\alpha| \cdot \|x\|$$
$$\|x + y\| \leq \|x\| + \|y\|$$

$$y_p(x) = \sum_{i=1}^{k} \omega_i y_i$$

$$\omega_i = \frac{(d_i + \frac{smear}{k})^{-1}}{\hat{d}}$$

$$\hat{d} = \sum_{i=1}^{k} \left(d_i + \frac{smear}{k}\right)^{-1}$$

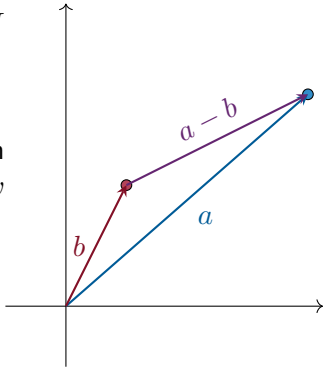


# Es geht darum, Abbildungen zwischen Vektorräumen zu lernen

- Wie schon erwähnt, geht es beim überwachten Lernen darum, eine Abbildung $f : X \to Y$ vom $m$-dimensionalen Merkmalsraum $X$ in einen $n$-dimensionalen Out-Raum $Y$ zu lernen.
- Nach einem Konvertierungsprozess stellen sich beide für den Computer als Teilmengen des $\mathbb{R}^n$ bzw. $\mathbb{R}^m$ dar.
- Natürlich stammen einige Merkmale ggf. aus Skalenniveaus mit weniger Struktur, und wir müssen vorsichtig sein.
- Nehmen wir einmal an, alle Merkmale entstammen einer rationalen Skala, und wir können die gleichen Operationen auf jedem Merkmal durchführen wie auf $\mathbb{R}$.
- In diesem Fall
$$f : \mathbb{R}^m \supset X \to Y \subset \mathbb{R}^n$$
ergibt sich vieles direkt aus den Eigenschaften von Vektorräumen, wie sie in der linearen Algebra besprochen werden.
- **Achtung**: Es geht nur um die Räume! $f$ ist im Allgemeinen nichtlinear.

# Wiederholung: Normen und Metriken

- Etwas verallgemeinert kann man sagen, dass Metriken dazu da sind, Abstände zwischen zwei Elementen $a, b$ in einer Menge $M$ zu bestimmen:
$$d(a, b) \to \mathbb{R}$$
- Auf einem normierten Vektorraum $X$ haben wir zusätzlich noch das Konzept der Norm. Eine Norm $\|.\|$ weist einem Vektor $v$ seine Länge zu:
$$\|v\| \to \mathbb{R}$$
- Beide Konzepte hängen zusammen: Norm induziert Metrik.
$$d(a, b) = \|a - b\|$$
- Zur Erinnerung: Das bedeutet, wenn man eine Norm hat, kann man darüber eine Metrik konstruieren. Es gibt aber Metriken, denen keine Normen zugrunde liegen.

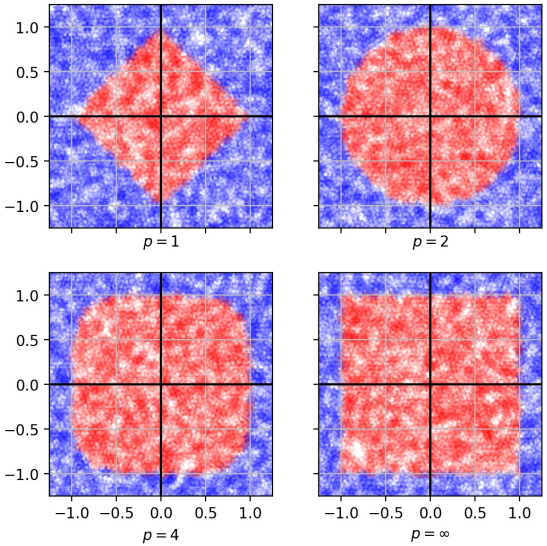


# $p$-Normen

- Wir nutzen Metriken und Normen, um die Ähnlichkeit zwischen zwei Objekten zu quantifizieren.
- Eine wichtige Klasse sind die $p$-Normen:

$$\|x\|_p = \left( \sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

- Rechts sehen Sie jeweils das Gebiet in rot illustriert, das in der jeweiligen $p$-Norm einen Abstand kleiner 1 vom Ursprung hat.



# Auszug aus einer Datenbank

| Nr. | Bezeichnung | 0<br>Preis € | 1<br>kW | 2<br>Hubraum | 3<br>kg | 4<br>l/100km | 5<br>Türen | Klasse |
|---|---|---|---|---|---|---|---|---|
| 2 | *Bugatti Chiron* | *2856000* | *1103* | *7993* | *2070* | *22.5* | *2* | *6* |
| 338 | Ford GT | 500000 | 475 | 3497 | 1385 | 14.9 | 2 | 6 |
| 361 | Lamborghini Urus | 204000 | 478 | 3996 | 2200 | 12.7 | 5 | 6 |
| 389 | Porsche 911 | 152416 | 368 | 3996 | 1488 | 12.9 | 2 | 6 |
| 126 | BMW M3 | 77500 | 317 | 2979 | 1595 | 8.8 | 4 | 4 |
| 145 | Alfa Romeo 4C | 63500 | 177 | 1742 | 970 | 6.8 | 2 | 4 |
| 308 | Porsche 718 | 52694 | 220 | 1988 | 1410 | 7.4 | 2 | 5 |
| 325 | Mercedes E 200 | 43019 | 135 | 1991 | 1575 | 6.1 | 4 | 5 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 40 | Opel Corsa OPC | 24930 | 152 | 1598 | 1293 | 7.5 | 3 | 2 |
| 251 | Toyota Avensis | 24740 | 97 | 1598 | 1430 | 6.1 | 4 | 4 |
| 512 | BMW 116i | 24700 | 80 | 1499 | 1375 | 5.3 | 3 | 3 |
| 41 | Peugeot 208 | 23990 | 153 | 1598 | 1235 | 5.4 | 3 | 2 |
| 9 | VW up! GTI | 16975 | 85 | 999 | 1070 | 4.8 | 3 | 1 |
| 591 | Toyota Auris 1.33 | 16490 | 73 | 1329 | 1225 | 5.5 | 5 | 3 |

# Ähnlichkeit über eine Metrik

## Wichtiger Hinweis

Es ist fast nie eine gute Idee, z. B. eine $p$-Metrik direkt auf einen Datensatz anzuwenden. Man muss sich erst die unterschiedlichen Größenordnungen der Merkmale klar machen.

| | Marke | Modell | Preis | Leist. | Hubr. | Gewicht | Verbrauch | Türen |
|---|---|---|---|---|---|---|---|---|
| 29 | Citroen | C1 VTi 68 | 9090 | 51 | 998 | 915 | 4.1 | 3 |
| 540 | Toyota | Corolla 1.6 | 21220 | 97 | 1598 | 1270 | 6.0 | 4 |

$$\left\| \begin{pmatrix} 9090 \\ 51 \\ 998 \\ 915 \\ 4.1 \\ 3 \end{pmatrix} - \begin{pmatrix} 21220 \\ 97 \\ 1598 \\ 1270 \\ 6.0 \\ 4 \end{pmatrix} \right\|_1 = 12130 + 46 + 600 + 355 + 1.9 + 1 = 13133.9 \text{ (Anteil Preis: } 92.4\%)$$

- Unskaliert kommen Merkmale mit geringer absoluter Streuung kaum zur Geltung.

# Relative und absolute Streuungsmaße

- Mittels Streuungsmaßen versucht man, die Streubreite von Werten in einer Datenmenge zu beschreiben.
- Dies geschieht immer relativ zu einem geeigneten Referenzpunkt. Bei der Gaußverteilung ist das der Mittelwert $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$.
- Das Streumaß ist hier entsprechend die Standardabweichung $s = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2}$.
- Es gibt auch Alternativen, wie z. B. die mittlere absolute Abweichung

$$\mathrm{e} = \frac{1}{n} \sum_{i=1}^{n} |x_i - \overline{x}|$$

- Nutzen wir nur die absoluten Werte, so handelt es sich um **absolute Streuungsmaße**.
- Normieren wir die Daten vorher, so sind es **relative Streuungsmaße**.

# Median und Perzentile

- Der Mittelwert stellt bei inhomogenen Gruppen die Menge oft ungenügend dar.

| 1.793 Euro | 1.979 Euro | 2.029 Euro | 2.157 Euro | 2.567 Euro | 5.400 Euro | 6.500 Euro |
|---|---|---|---|---|---|---|
| Haushaltshilfe | Empfangskraft | Koch/Köchin | Arzthelfer/-in | Maler/-in | Informatiker/-in | Wirtschaftsprüfer/-in |

- Bei der oben angeben Gruppe beträgt der Mittelwert `np.mean(X)` $= 3203.57\ldots$ Euro, der Median hingegen `np.median(X)` $= 2157$ Euro $=$ `np.percentile(X,50)`.
- Der Hilfe von `np.percentile` können auch entsprechenden Zwischengrößen angefragt werden `np.percentile(X,25)` $= 2004.0$ Euro.
- Im Allgemeinen ist der Median robuster gegenüber Ausreißern als der Mittelwert.

# Skalierung

Um den Merkmalen einen ähnlichen Einfluss auf den Abstand zu gewähren, können wir die Daten spaltenweise

- normieren: $\hat{x} = \frac{x - \min(x)}{\max(x) - \min(x)}$

```
Xmin = X.min(axis=0)
Xmax = X.max(axis=0)
X = (X - Xmin) / (Xmax - Xmin)
```

- standardisieren: $\breve{x} = \frac{x - \bar{x}}{\sigma_x}$ mit $\bar{x} = \frac{1}{n}\sum x_i$ und $\sigma_x = \sqrt{\frac{1}{n}\sum (x_i - \bar{x})^2}$

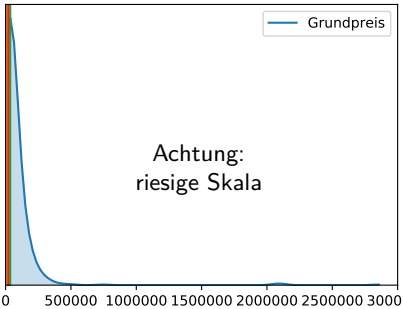
```
Xmean = X.mean(axis=0)
Xstd = X.std(axis=0)
X = (X - Xmean) / Xstd
```

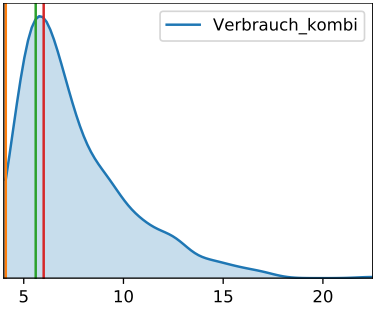
- robust skalieren: $\breve{x} = \frac{x - x_{p_{50}}}{x_{p_{75}} - x_{p_{25}}}$

```
X25 = np.percentile(X, q=25, axis=0)
X50 = np.percentile(X, q=50, axis=0)
X75 = np.percentile(X, q=75, axis=0)
X = (X - X50) / (X75 - X25)
```

# Unskaliert (i.W. Rohdaten)

| | Marke | Modell | Preis | Leist. | Hubr. | Gewicht | Verbrauch | Türen |
|---|---|---|---|---|---|---|---|---|
| 29 | Citroen | C1 VTi 68 | 9090 | 51 | 998 | 915 | 4.1 | 3 |
| 200 | Mercedes | SLC 180 | 35349 | 115 | 1595 | 1435 | 5.6 | 2 |
| 540 | Toyota | Corolla 1.6 | 21220 | 97 | 1598 | 1270 | 6.0 | 4 |



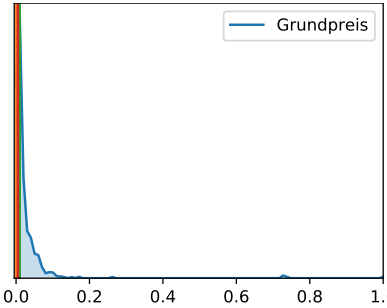



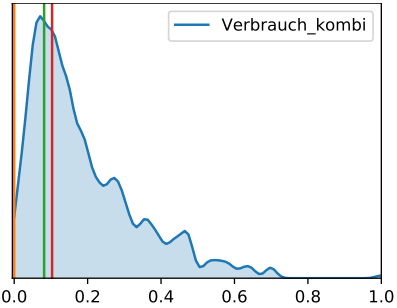

Abstände (1-Norm) und der Beitrag des Preises daran:

- Auto 29 zu Auto 200: $27442.5$ (Anteil Preis: 95.7%)
- Auto 29 zu Auto 540: $13133.9$ (Anteil Preis: 92.4%)
- Auto 200 zu Auto 540: $14317.4$ (Anteil Preis: 98.7%)

# Normiert

| | Marke | Modell | Preis | Leist. | Hubr. | Gewicht | Verbrauch | Türen |
|---|---|---|---|---|---|---|---|---|
| 29 | Citroen | C1 VTi 68 | 0.0007 | 0.0066 | 0.0141 | 0.0946 | 0.0000 | 0.3333 |
| 200 | Mercedes | SLC 180 | 0.0100 | 0.0670 | 0.0982 | 0.3181 | 0.0815 | 0.0000 |
| 540 | Toyota | Corolla 1.6 | 0.0050 | 0.0500 | 0.0987 | 0.2472 | 0.1033 | 0.6667 |



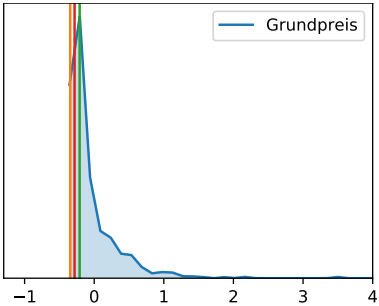



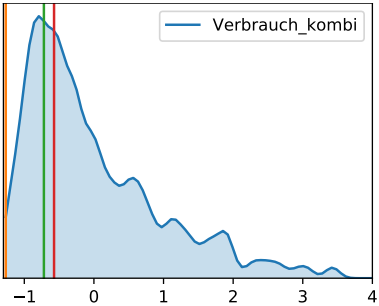

Abstände (1-Norm) und der Beitrag des Preises daran:

- Auto 29 zu Auto 200: $0.7922$ (Anteil Preis: 1.1%)
- Auto 29 zu Auto 540: $0.7215$ (Anteil Preis: 0.6%)
- Auto 200 zu Auto 540: $0.7817$ (Anteil Preis: 0.6%)

# Standardisiert

| | Marke | Modell | Preis | Leist. | Hubr. | Gewicht | Verbrauch | Türen |
|---|---|---|---|---|---|---|---|---|
| 29 | Citroen | C1 VTi 68 | -0.3445 | -1.0134 | -0.9692 | -1.7027 | -1.2661 | -0.7933 |
| 200 | Mercedes | SLC 180 | -0.2092 | -0.5520 | -0.5543 | -0.2871 | -0.7186 | -1.6091 |
| 540 | Toyota | Corolla 1.6 | -0.2820 | -0.6817 | -0.5522 | -0.7363 | -0.5726 | 0.0226 |



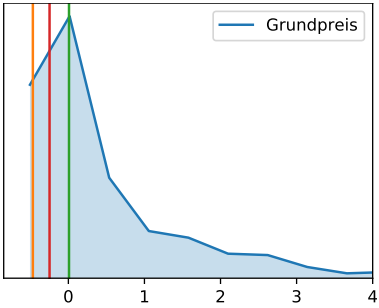



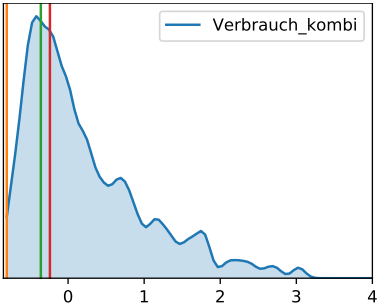

Abstände (1-Norm) und der Beitrag des Preises daran:

- Auto 29 zu Auto 200: $3.7906$ (Anteil Preis: 3.6%)
- Auto 29 zu Auto 540: $3.2869$ (Anteil Preis: 1.9%)
- Auto 200 zu Auto 540: $2.4315$ (Anteil Preis: 3%)

# Robust Skaliert

| | Marke | Modell | Preis | Leist. | Hubr. | Gewicht | Verbrauch | Türen |
|---|---|---|---|---|---|---|---|---|
| 29 | Citroen | C1 VTi 68 | -0.4668 | -0.4677 | -0.6058 | -1.2513 | -0.8060 | -1.0 |
| 200 | Mercedes | SLC 180 | 0.0084 | -0.1237 | -0.2390 | -0.1390 | -0.3582 | -1.5 |
| 540 | Toyota | Corolla 1.6 | -0.2473 | -0.2204 | -0.2372 | -0.4920 | -0.2388 | -0.5 |




Abstände (1-Norm) und der Beitrag des Preises daran:

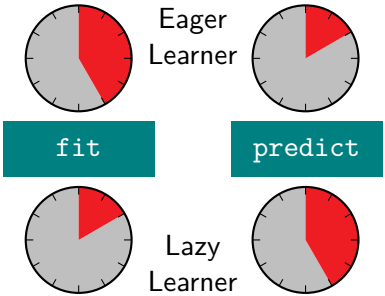
- Auto 29 zu Auto 200: $3.2462$ (Anteil Preis: 14.6%)
- Auto 29 zu Auto 540: $2.662$ (Anteil Preis: 8.2%)
- Auto 200 zu Auto 540: $1.8266$ (Anteil Preis: 14%)

# Fazit zum Preprocessing

- Die Normierung kann stark von Ausreißern beeinflusst werden. Die Standardisierung und die robuste Skalierung sind weniger anfällig. Generell sollte das bewusste und dokumentierte Entfernen von Ausreißern erwogen werden.
- Die Normierung hat den Vorteil, dass der Wertebereich wie z. B. $[0, 1]$ auch eingehalten wird, was manchmal nützlich ist.
- Anstatt Standardisierung müsste der Ansatz eigentlich Studentisierung heißen, weil er auf der empirischen Standardabweichung basiert, aber es ist üblich, den Ansatz unpräzise als Standardisierung zu bezeichnen.
- Nach einer Standardisierung beträgt der Mittelwert 0 und die Standardabweichung (und damit auch die Varianz) 1.
- Merkmale lassen sich auch bewusst skalieren, um ihnen einen größeren Anteil beim Abstand zukommen zu lassen.
- Viele Verfahren basieren auf Abständen. Ohne Skalierung funktionieren sie oft nicht.
- Neuronale Netze können ohne **Preprocessing** der Daten Konvergenzprobleme haben.

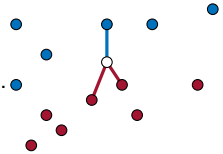
# Eager Learning vs. Lazy Learning

- Die meisten Algorithmen im Bereich des maschinellen Lernens basieren auf einem Ansatz, den man **Eager Learning** nennt.
- Hierbei wird der Hauptteil der Arbeit während des Trainings investiert. Ein typisches Beispiel für ein solches Verfahren sind neuronale Netze.
- Der **k-Nearest Neighbors** (k-NN) Algorithmus ist hingegen ein **Lazy Learner**.
- Bei dieser Klasse von Verfahren findet die Hauptarbeit nicht beim Training statt, sondern erst zur Zeit der Anfrage.



# Grundidee des k-NN

- Die Grundidee beim k-NN ist es, eine Regression oder Klassenzuordnung für einen Abfragepunkt $x$ auf der Basis seiner $k$ nächsten Nachbarn durchzuführen.
- Nehmen wir als Beispiel die Klassifikation. Hierbei bestehen die Arbeitsschritte aus:
  1. Bestimme die Distanz in der Metrik $d$ von $x$ zu allen Samples.
  2. Finde die $k$ Samples, deren Distanz am gerinsten ist (nächste Nachbarn).
  3. Liefere die häufigste Klasse als Ergebnis der Klassifikation zurück.

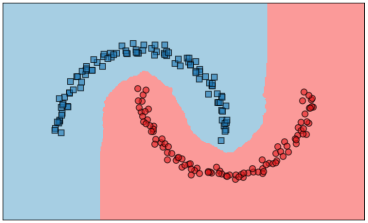


- Die meiste Arbeit steckt in den Schritten 1 und 2, da wir hier von allen Elementen unserer Datenbank die Distanz zu $x$ bestimmen und die $k$ geringsten Werte finden müssen.
- Statt einfach nur abzuzählen, was die häufigste Klasse ist, kann man auch jeden Nachbarn noch gewichten, z. B. auf der Basis der jeweiligen Distanz.
- Ein Vorteil des k-NN für manche Anwendungen ist, dass er mit einem lokalen Ansatz arbeitet und nicht auf ein globales Modell angewiesen ist.

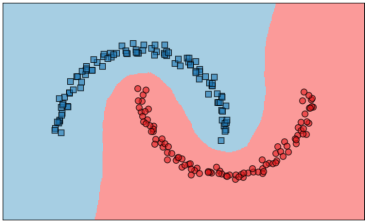
# Einfluss der Norm am Beispiel des Two Moons Set

- Rechts sehen Sie die Klassifikation mit k-NN für unterschiedliche Normen.
- Neben $k$ – hier 5 – und der Gewichtung der Nachbarn ist die verwendete Norm der wichtigste **Hyperparameter**.
- Hyperparameter sind Parameter, die fest gewählt werden und beim Training von den Daten nicht beeinflusst werden.

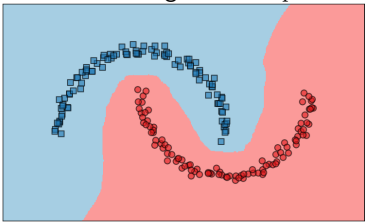
$k$ Nearest Neighbors mit $p = 1$



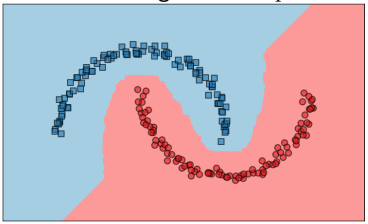
$k$ Nearest Neighbors mit $p = 2$



$k$ Nearest Neighbors mit $p = 4$



$k$ Nearest Neighbors mit $p = \infty$



# Gewichtung bei der Regression

- Bei der Regression geht man analog vor, hier ist jedoch die Gewichtung noch wesentlicher. Auch müssen hier zwingend alle Gewichte addiert 1 ergeben.
- Klassisch sähe eine reine Gewichtung über die Abstände $d_i = d(x_i, x)$ so aus:

$$y_p(x) = \frac{1}{\sum_{i=1}^{k} \hat{\omega}_i} \sum_{i=1}^{k} \hat{\omega}_i y_i \quad \text{mit} \quad \hat{\omega}_i = \frac{1}{d_i}$$

- Das berücksichtigt weder den Fall einer Division durch Null noch fehlerhafter Werte.
- Liegt ein Nachbar wesentlich näher am Abfragepunkt als die anderen, dominiert er den Wert. Das ist gut, wenn die Werte in der Datenbank fehlerfrei sind; jedoch schlecht, wenn wir davon ausgehen, Fehler herausmitteln zu müssen.
- Man greift hier oft dazu die Gewichte etwas zu "verschmieren":

$$y_p(x) = \frac{1}{\sum_{i=1}^{k} \omega_i} \sum_{i=1}^{k} \omega_i y_i \quad \text{mit} \quad \omega_i = \frac{1}{d_i + \varepsilon}$$

# k-NN mit Suchbaum Unterstützung

- Eine Schwäche des Verfahrens ist der Aufwand, der pro Abfrage entsteht.
- Der Aufwand für die Berechnung der Abstände wächst linear mit der Anzahl der Elemente im Trainingsset.
- Der Aufwand für die Sortierung wächst für die meisten Algorithmen mit $O(n \cdot \log(n))$.
- Das Problem ist, dass dieser Aufwand pro Anfrage anfällt.
- Der Ausweg ist eine Art Mischansatz, in dem man etwas Arbeit zur Trainingsphase investiert, um die Trainingsdaten so zu organisieren.
- Auf diesen organisierten Daten ist es dann möglich, die Suche nach Nachbarn nur noch auf einer Teilmenge durchzuführen.
- Eine der in der Praxis gut bewährten Methoden ist der Einsatz eines $k$-dimensionalen Suchbaums, kurz **kd-Baum** bzw. **kd-tree**.
- Die Effizienz dieses kd-trees hängt von der Anzahl der Dimensionen bzw. Merkmale ab. Weniger ist besser.