Machine Learning

Prof. Dr. Fabian Brunner

<fa.brunner@oth-aw.de>

Amberg, 12. Oktober 2021

Themen und Lernziele heute

- Formulierung von Klassifikationsproblemen
- Kategorisierung von Machine Learning-Verfahren
- Funktionsweise von kNN
- Bewertung der Zeitkomplexität von ML-Verfahren am Beispiel von kNN
- Effiziente Implementierung von kNN mit Hilfe von k-d-Bäumen
- Implementierung von kNN in Python
- Anwendung von kNN in der ML-Bibliothek sklearn

Klassifikationsprobleme - allgemeine Notation

Klassifikationsproblem:
Wir betrachten im Folgenden stets Klassifikationsprobleme, bei denen eine gegebene Instanz anhand von

- p numerischen Attributen (bzw. Features)
- einer der Klassen $C_1, \ldots, C_n$

zugeordnet werden soll. Für $n = 2$ spricht man von **binärer Klassifikation**.
Gesucht ist also eine Abbildung

$$f : \mathbb{R}^p \to \{C_1, \ldots, C_n\} \ ,$$

die anhand gegebener Trainingsdaten geeignet zu bestimmen ist.

Trainingsdaten für Supervised Learning:
Dazu sei ein Trainingsdatensatz bestehend aus m Samples

$$(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)})$$

mit Labels/Zielfunktionswerten gegeben, wobei $\boldsymbol{x}^{(i)} \in \mathbb{R}^p$ und $y^{(i)} \in \{C_1, \ldots, C_n\}$.

Beispiel: Klassifikation von Schwertlilien

- Die **Iris-Datensammlung** ist ein im Jahr 1936 vom Biologen Ronald Fisher veröffentlichter Datensatz, der häufig als Beispieldatensatz verwendet wird.
- Er enthält je 50 Samples zu den drei Schwertlilien-Arten Iris Setosa, Iris Versicolor und Iris Virginica.
- Jedes Sample besteht aus vier Angaben/Features:
  - Länge des Blütenblatts (petal length)
  - Breite des Blütenblatts (petal width)
  - Länge des Kelchblatts (sepal length)
  - Breite des Kelchblatts (sepal width)

Klassifikationproblem

Ordne die Schwertlilien anhand der vier Features **petal length, petal width, sepal length, sepal width** der richtigen von drei möglichen Klassen (setosa, versicolor, virginica) zu

Iris-Datensammlung

iris setosa

Blütenblatt (petal)

Kelchblatt (sepal)

iris versicolor

Blütenblatt (petal)

Kelchblatt (sepal)

iris virginica

Blütenblatt (petal)

Kelchblatt (sepal)

Aufbau des Datensatzes

Nr.	sepal length	sepal width	petal length	petal width	Klasse
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
⋮					⋮
51	7.0	3.2	4.7	1.4	Versicolor
52	6.4	3.2	4.5	1.5	Versicolor
⋮					⋮
101	6.3	3.3	6.0	2.5	Virginica
102	5.8	2.7	5.1	1.9	Virginica
⋮					⋮
150	5.9	3.0	5.1	1.8	Virginica

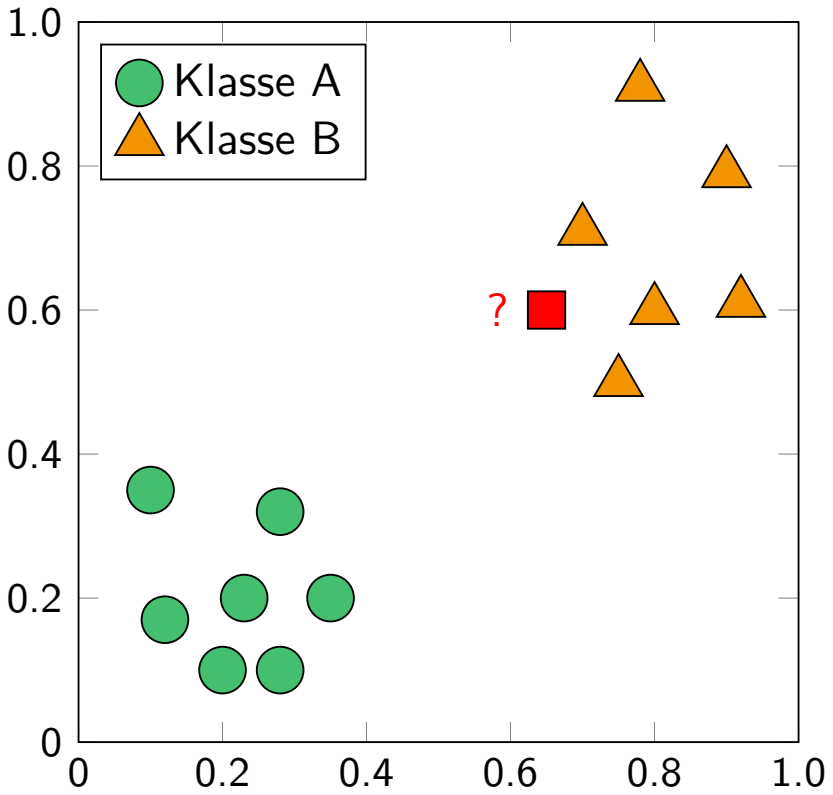
Features

Zielvariable

Nearest Neighbor-Klassifikation

Gegeben: Trainingsdatenpunkte im $\mathbb{R}^2$ aus zwei Klassen A und B wie in der folgenden Abbildung

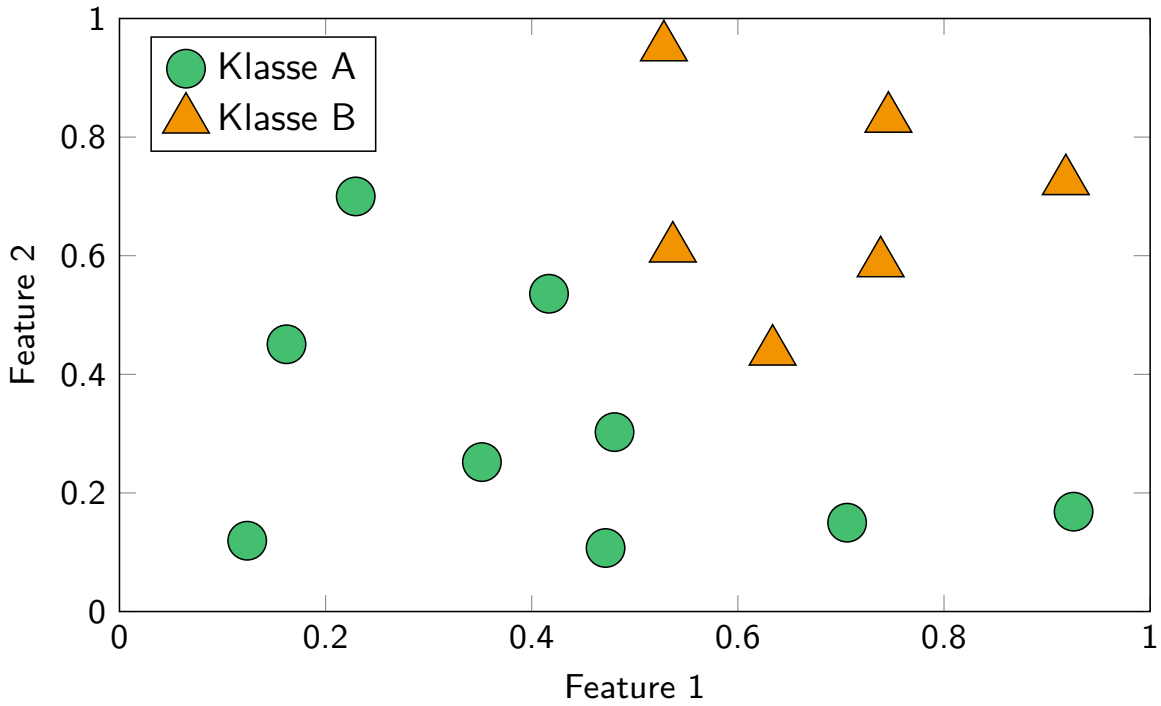
Frage: Gehört das rote Quadrat zur Klasse A oder B?



Idee der Nearest Neighbor-Klassifikation

Grundidee der Nearest Neighbor-Klassifikation

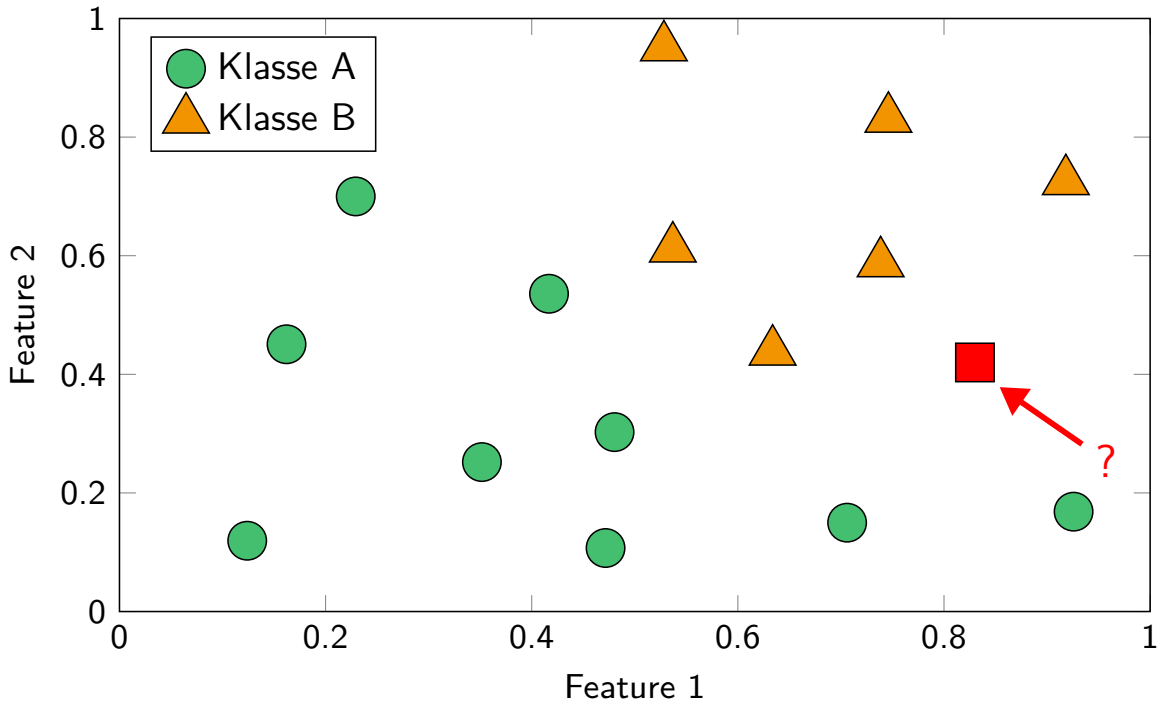
- Um die Klasse eines neuen, unbekannten Objekts vorherzusagen, vergleicht man das zu klassifizierende Objekt mit den Beispielen aus dem Trainingsdatensatz und ordnet ihm dieselbe Klasse zu wie der Instanz aus dem Trainingsdatensatz, die dem Objekt am ähnlichsten ist.
- Zur Messung der Ähnlichkeit verwendet man ein Ähnlichkeitsmaß, z.B. den Euklidischen Abstand.



Idee der Nearest Neighbor-Klassifikation

Grundidee der Nearest Neighbor-Klassifikation

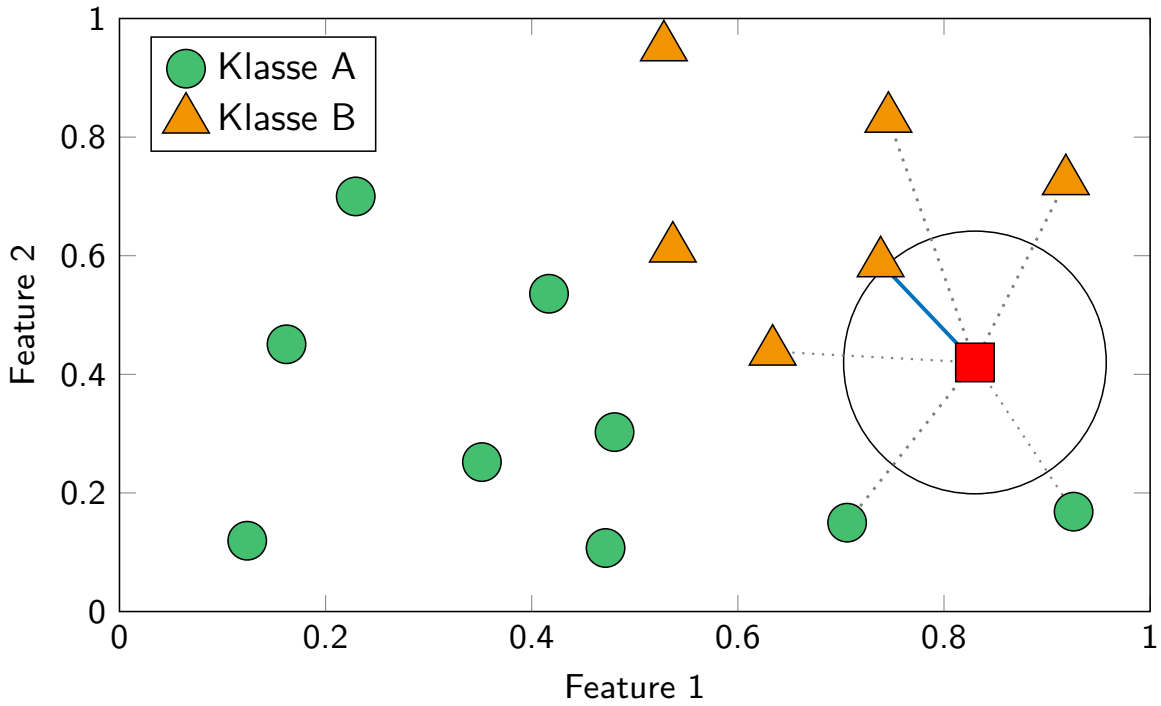
- Um die Klasse eines neuen, unbekannten Objekts vorherzusagen, vergleicht man das zu klassifizierende Objekt mit den Beispielen aus dem Trainingsdatensatz und ordnet ihm dieselbe Klasse zu wie der Instanz aus dem Trainingsdatensatz, die dem Objekt am ähnlichsten ist.
- Zur Messung der Ähnlichkeit verwendet man ein Ähnlichkeitsmaß, z.B. den Euklidischen Abstand.



Idee der Nearest Neighbor-Klassifikation

Grundidee der Nearest Neighbor-Klassifikation

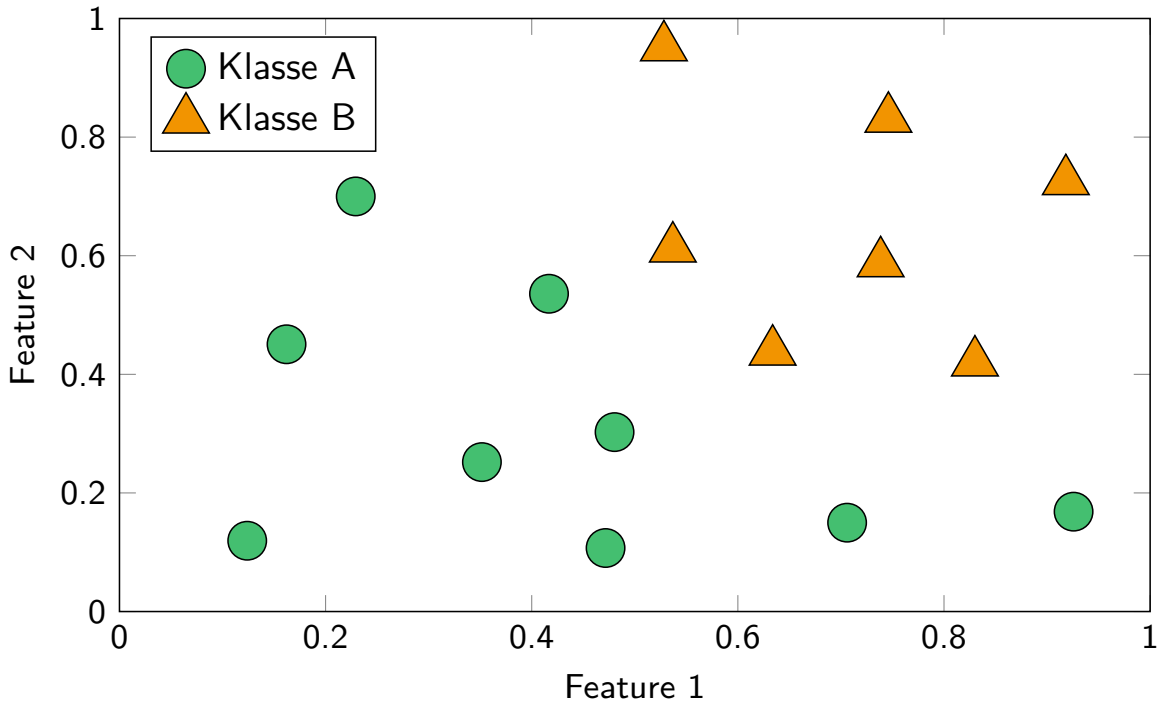
- Um die Klasse eines neuen, unbekannten Objekts vorherzusagen, vergleicht man das zu klassifizierende Objekt mit den Beispielen aus dem Trainingsdatensatz und ordnet ihm dieselbe Klasse zu wie der Instanz aus dem Trainingsdatensatz, die dem Objekt am ähnlichsten ist.
- Zur Messung der Ähnlichkeit verwendet man ein Ähnlichkeitsmaß, z.B. den Euklidischen Abstand.



Idee der Nearest Neighbor-Klassifikation

Grundidee der Nearest Neighbor-Klassifikation

- Um die Klasse eines neuen, unbekannten Objekts vorherzusagen, vergleicht man das zu klassifizierende Objekt mit den Beispielen aus dem Trainingsdatensatz und ordnet ihm dieselbe Klasse zu wie der Instanz aus dem Trainingsdatensatz, die dem Objekt am ähnlichsten ist.
- Zur Messung der Ähnlichkeit verwendet man ein Ähnlichkeitsmaß, z.B. den Euklidischen Abstand.



Einordnung

Kategorisierung der Nearest Neighbor-Klassifikation:

- Da die Klassen im Trainingsdatensatz bekannt sind, handelt es sich bei der Nearest Neighbor-Klassifikation um ein Verfahren des **Supervised Learning**.
- Da die Klassifikation allein auf Grund eines Vergleichs mit den Trainingsinstanzen erfolgt, spricht man von **instanzbasiertem** Lernen bzw. **lazy learning**, s. übernächste Folie.
- Die Nearest Neighbor-Klassifikation ist ein Beispiel für ein **nicht-parametrisiertes** Verfahren (s. nächste Folie).

Nearest Neighbor-Klassifikation als nicht-parametrisiertes Modell

Ostbayerische Technische Hochschule Amberg-Weiden

Parametrisierte und nicht-parametrisierte Lernverfahren

- Bei parametrisierten Lernverfahren werden in der Trainingsphase anhand der Trainingsdaten die Parameter einer Funktion aus einer festgelegten Funktionsklasse bestimmt, die den Zusammenhang zwischen Eingabe- und Zielvariablen modelliert.
- Die Struktur der Funktion ist dabei fest vorgegeben und charakteristisch für das jeweilige Lernverfahren (z.B. lineare Funktion $\leftrightarrow$ lineare Regression).
- Nachdem die Parameter bestimmt wurden, kann die resultierende Funktion auf neue Daten angewendet werden, z.B. um Vorhersagen zu treffen.
- Im Gegensatz dazu wird bei nicht-parametrisierten Modellen eine Funktion gelernt, deren Struktur vor dem Training nicht konkret, z.B. durch eine bestimmte Anzahl an Parametern, festgelegt ist.
- Die Nearest Neighbor-Klassifikation ist ein nicht-parametrisiertes Lernverfahren.

Nearest Neighbor-Klassifikation als Beispiel für „Lazy Learning“

Ostbayerische Technische Hochschule Amberg-Weiden

„Lazy Learning“ und „Eager Learning“

- Die Nearest Neighbor-Klassifikation wird dem „**lazy learning**“ zugerechnet, bei dem die Trainingsdaten einfach abgespeichert werden und die Modellbildung online zum Zeitpunkt der Abfrage erfolgt.
- Im Gegensatz dazu erfolgt beim „**eager learning**“ die Modellbildung offline einmalig basierend auf den Trainingsdaten.

Verständnisfragen:

- Welche Nachteile hat „Lazy Learning“?
- In welchen Situationen kann „Lazy Learning“ vorteilhaft sein?

Pseudocode für die lineare Nearest Neighbor-Suche

Trainingsphase:

For i=1 to m:
store training example $(\boldsymbol{x}^{(i)}, y^{(i)})$

Klassifikation von x:

nn = None
dist_nn = ∞

for i=1 to m:

    dist:= $d(\boldsymbol{x}^{(i)}, \boldsymbol{x})$
if dist < dist_nn:
dist_nn = dist
nn = $\boldsymbol{x}^{(i)}$

return target value of nn

Komplexität

Welche Zeitkomplexität hat die (lineare) Nearest Neighbor-Klassifikation in Abhängigkeit von m (Anzahl der Datensätze) und p (Anzahl der Features)?

Komplexität

Welche Zeitkomplexität hat die (lineare) Nearest Neighbor-Klassifikation in Abhängigkeit von m (Anzahl der Datensätze) und p (Anzahl der Features)?

- Es müssen m Distanzberechnungen durchgeführt werden.
- Jede Distanzberechnung erfordert das Durchlaufen aller p Komponenten der Datenpunkte.

Zeitkomplexität der linearen Nearest Neighbor-Suche

$$\mathcal{O}(m \cdot p)$$

Ziel im Folgenden: Reduktion der Komplexität durch Nutzung geeigneter Datenstrukturen ($\rightarrow$ k-d-Bäume)

k-d-Baum

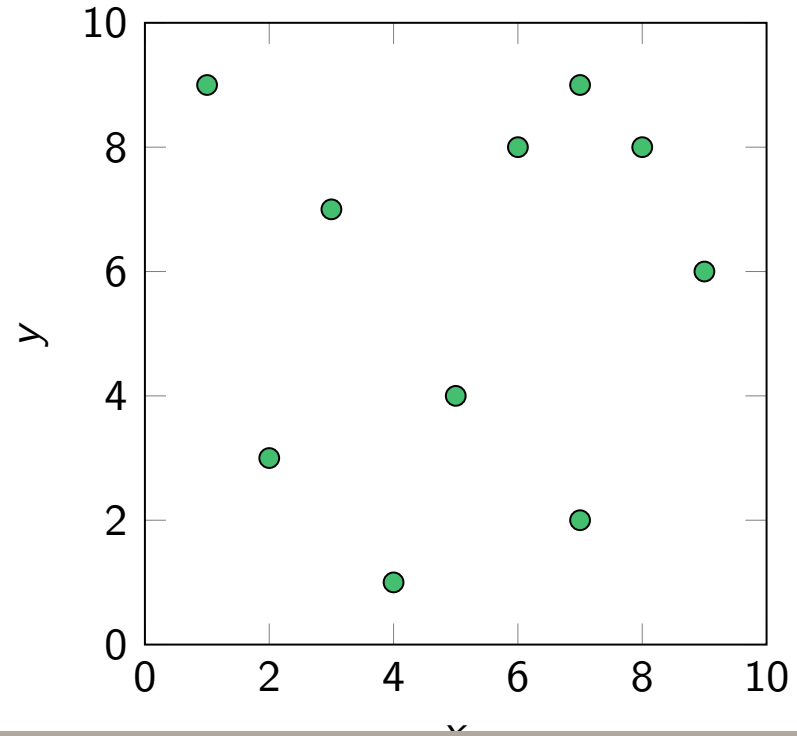
Idee: Hierarchische Speicherung der Trainingsdatenpunkte zur effizienteren Suche

Rekursive Konstruktion eines K-d-Trees

In jedem Knoten (falls mehr als `min_samples_leave` Samples vorhanden):

- Auswahl einer Dimension (z.B. zyklisch oder zufällig)
- Berechnung des Medians
- Split am Median

(1,9),(2,3),(4,1),(3,7),(5,4),(6,8),(7,2),(8,8),(7,9),(9,6)



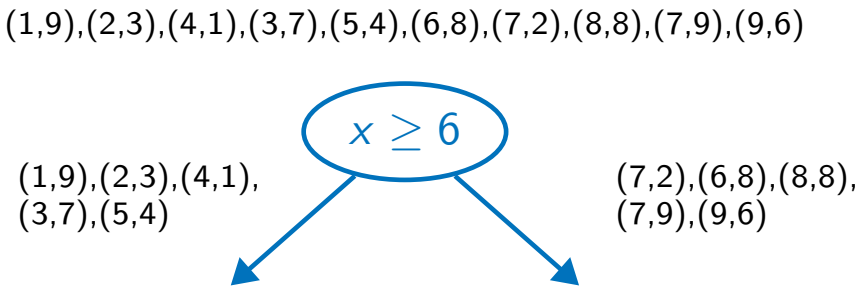
k-d-Baum

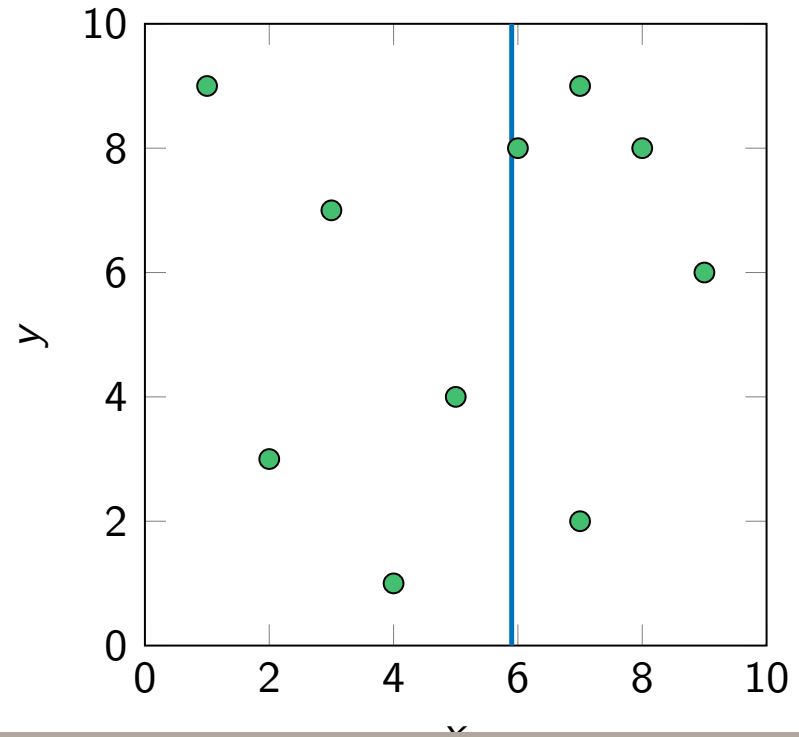
Idee: Hierarchische Speicherung der Trainingsdatenpunkte zur effizienteren Suche

Rekursive Konstruktion eines K-d-Trees

In jedem Knoten (falls mehr als `min_samples_leave` Samples vorhanden):

- Auswahl einer Dimension (z.B. zyklisch oder zufällig)
- Berechnung des Medians
- Split am Median





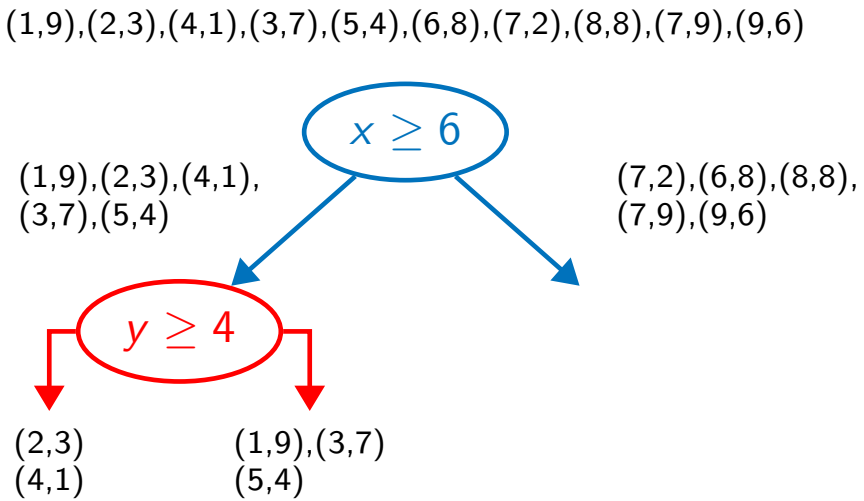
k-d-Baum

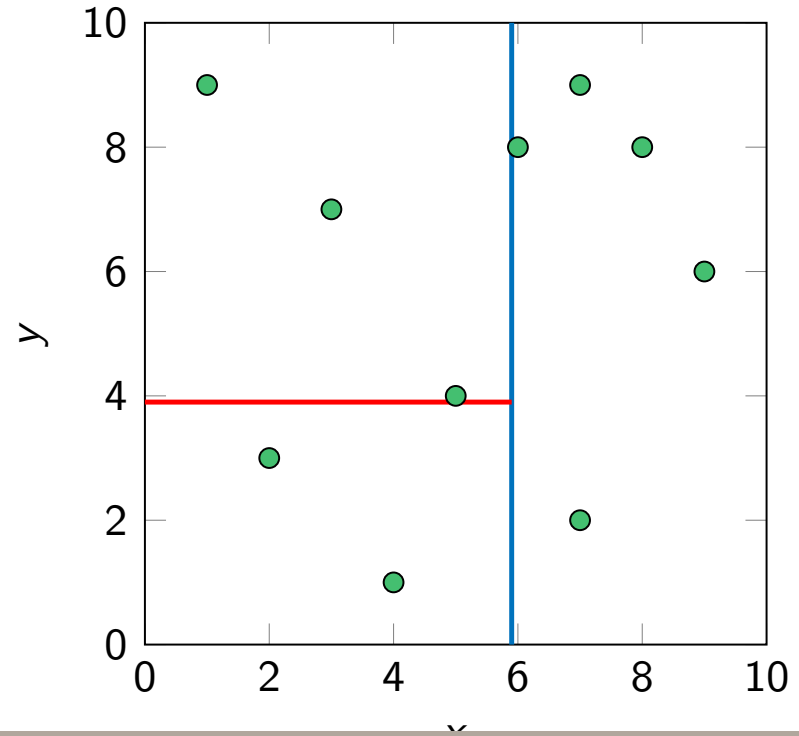
Idee: Hierarchische Speicherung der Trainingsdatenpunkte zur effizienteren Suche

Rekursive Konstruktion eines K-d-Trees

In jedem Knoten (falls mehr als `min_samples_leave` Samples vorhanden):

- Auswahl einer Dimension (z.B. zyklisch oder zufällig)
- Berechnung des Medians
- Split am Median





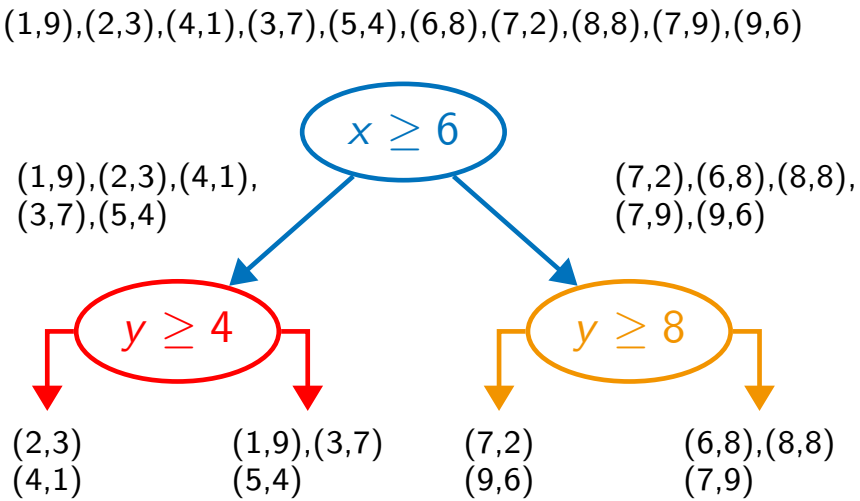
k-d-Baum

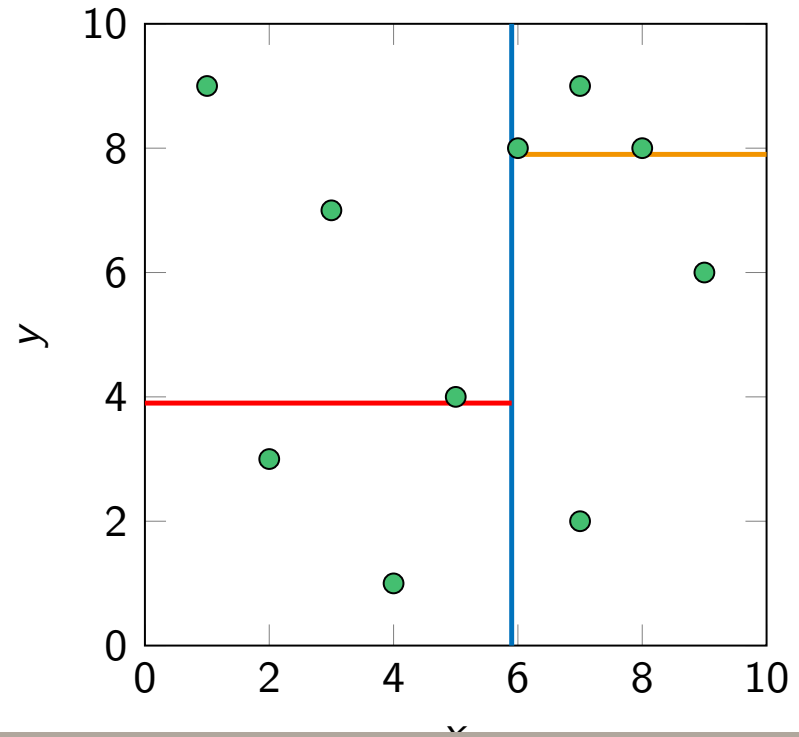
Idee: Hierarchische Speicherung der Trainingsdatenpunkte zur effizienteren Suche

Rekursive Konstruktion eines K-d-Trees

In jedem Knoten (falls mehr als `min_samples_leave` Samples vorhanden):

- Auswahl einer Dimension (z.B. zyklisch oder zufällig)
- Berechnung des Medians
- Split am Median





Komplexität der Konstruktion eines k-d-Baums

Der Median einer m-elementigen Menge kann mit einer Zeitkomplexität von $\mathcal{O}(m)$ berechnet werden.

Sei $T(m)$ die Zeitkomplexität für den Aufbau eines k-d-Baums mit m Punkten. Dann gilt:

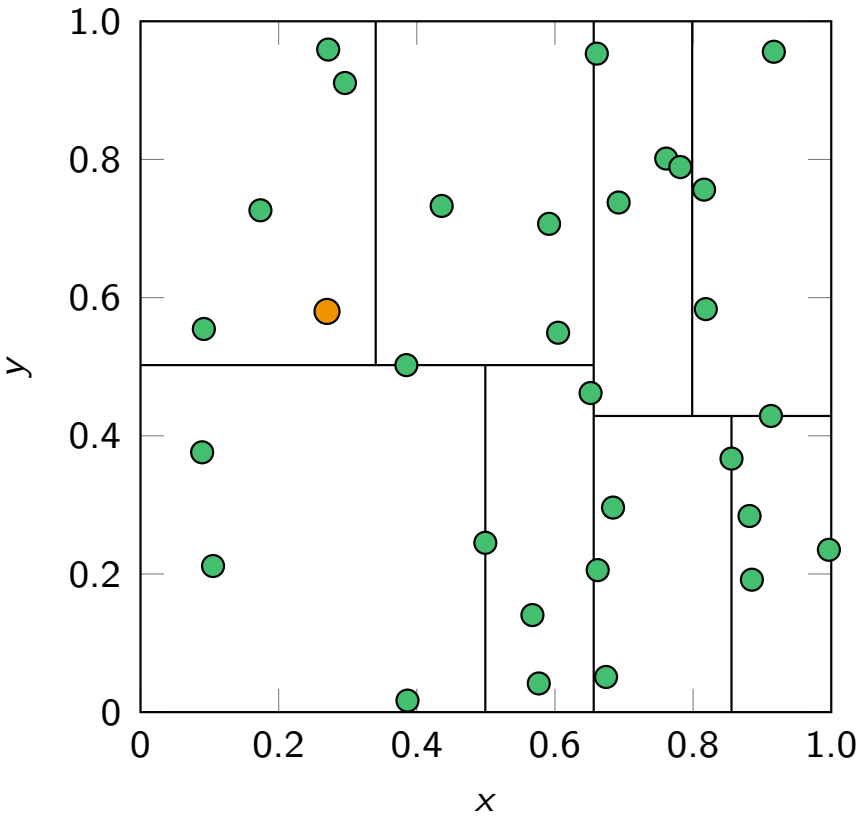
$$T(1) = 1,$$
$$T(m) = 2 \cdot T\left(\frac{m}{2}\right) + \mathcal{O}(m) \ .$$

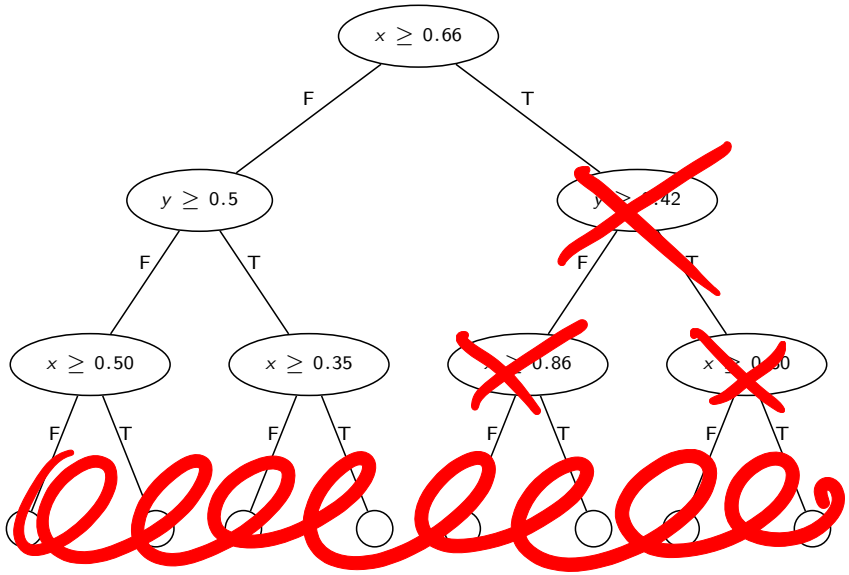
Fazit:

Zeitkomplexität der Konstruktion eines k-d-Baums mit m Punkten

$$\mathcal{O}(m \log m)$$

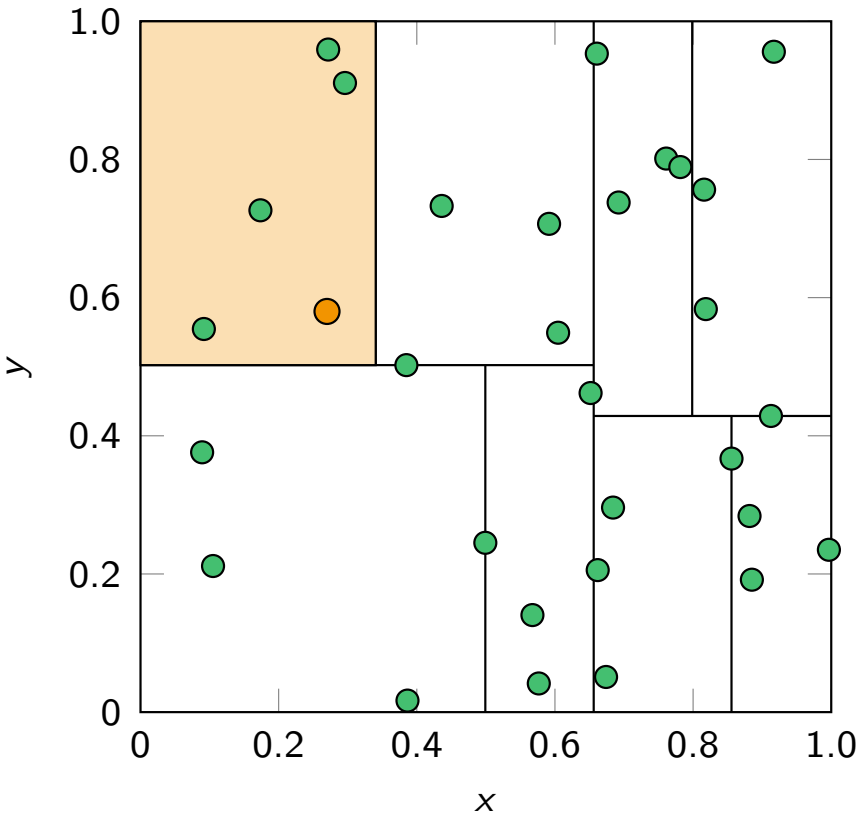
Nearest Neighbor Suche im k-d-Baum

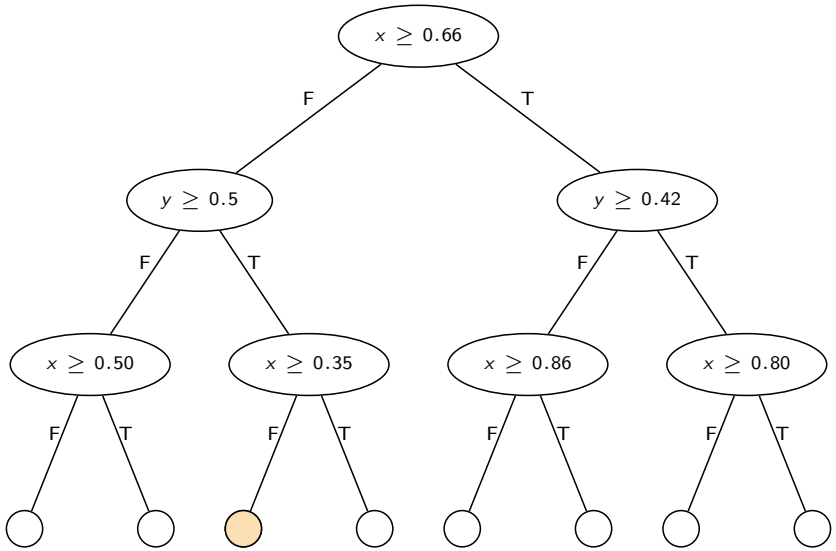




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

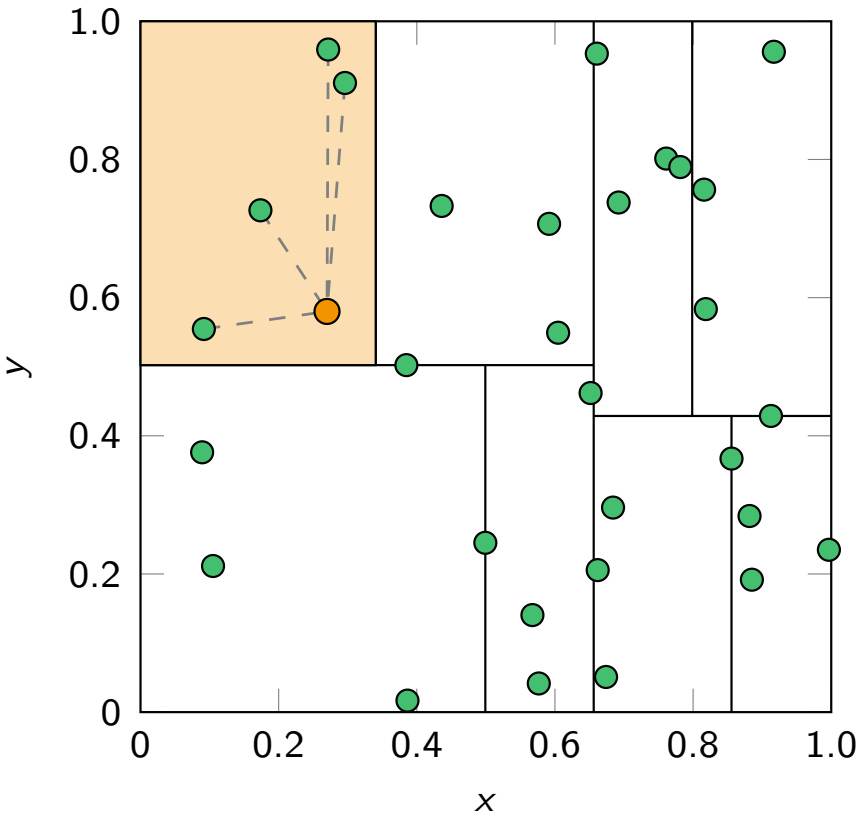
Nearest Neighbor Suche im k-d-Baum

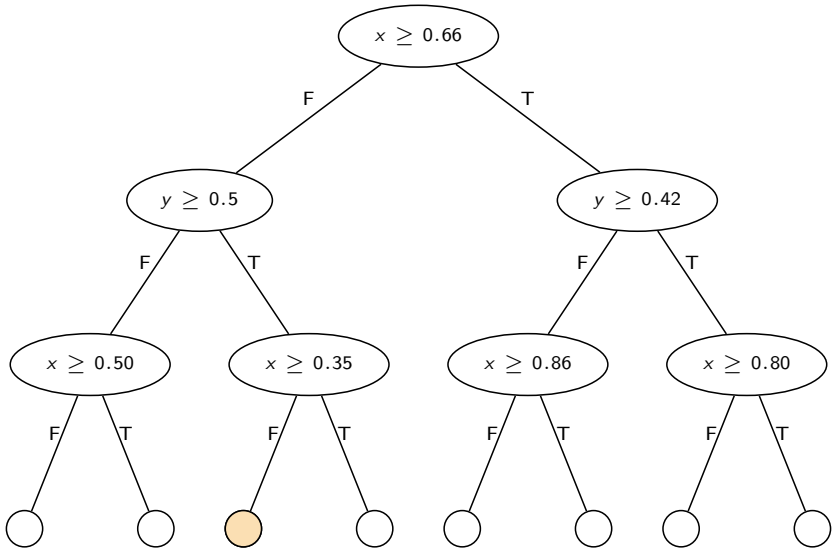




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

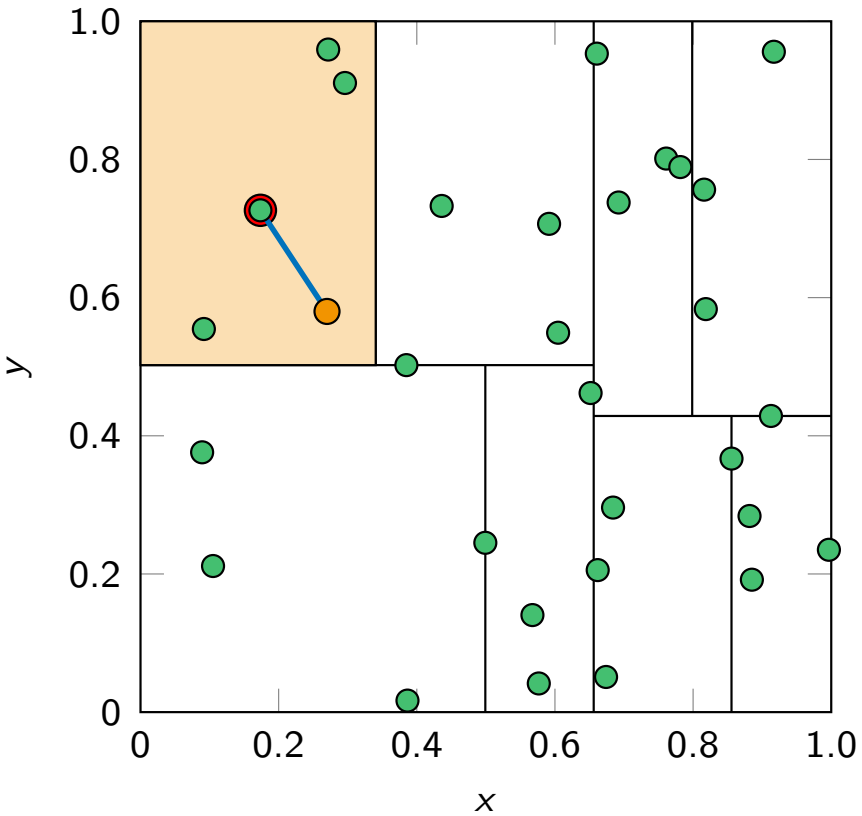
Nearest Neighbor Suche im k-d-Baum

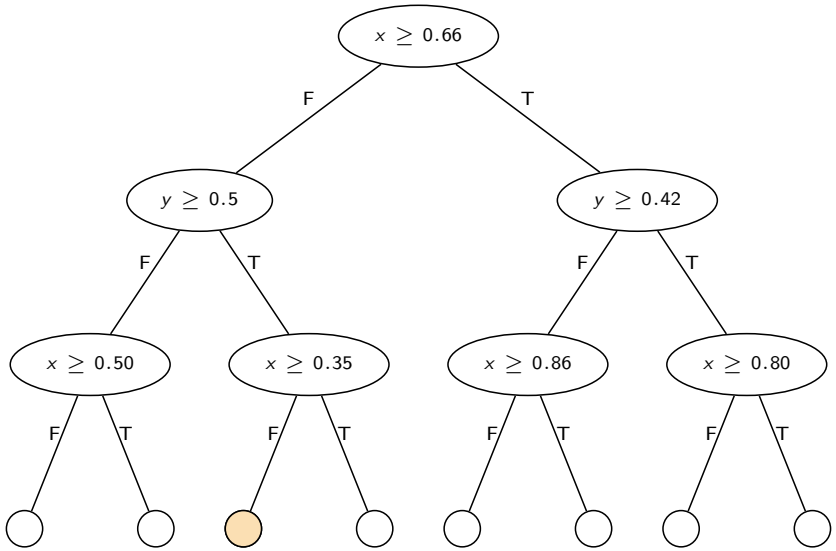




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

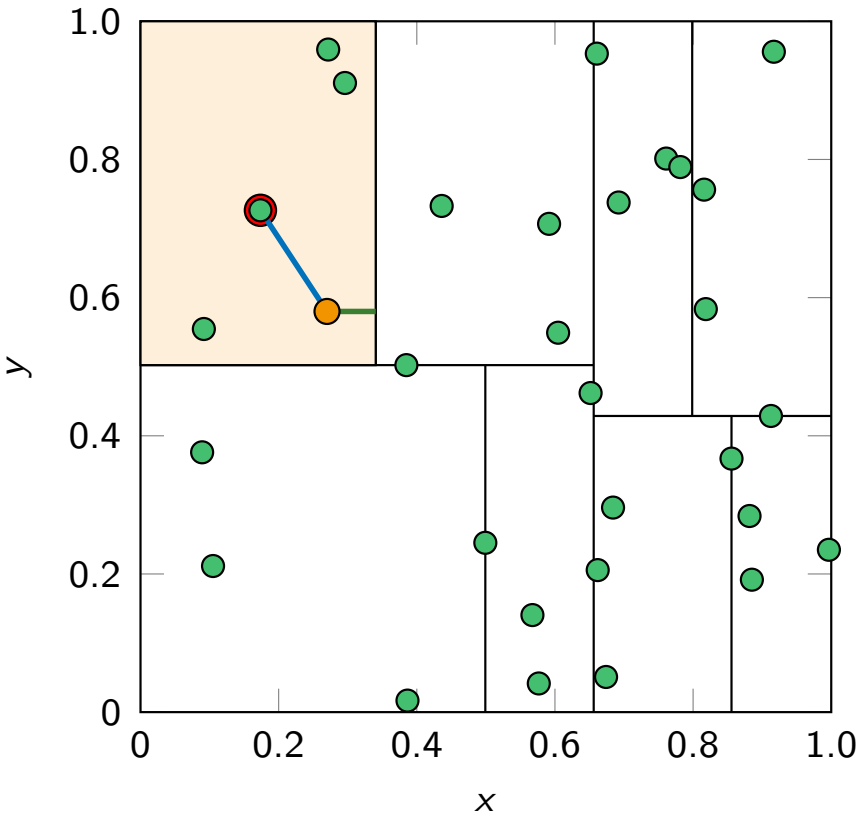
Nearest Neighbor Suche im k-d-Baum

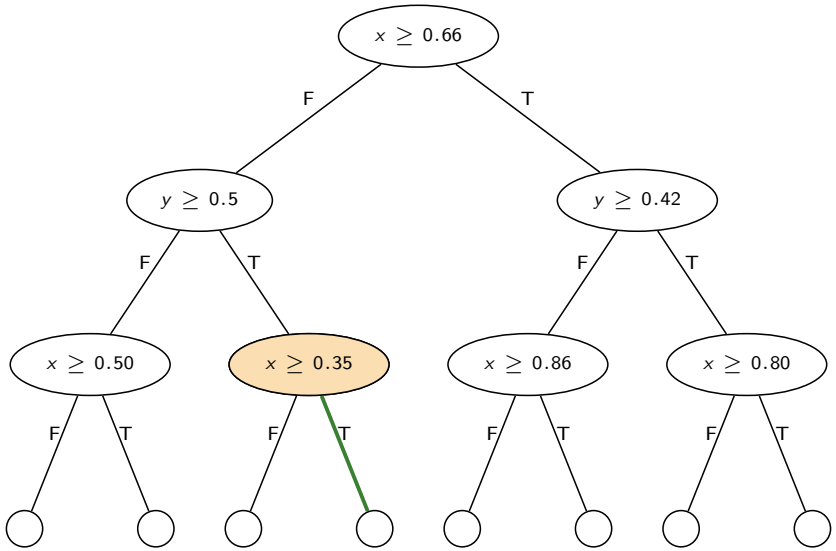




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

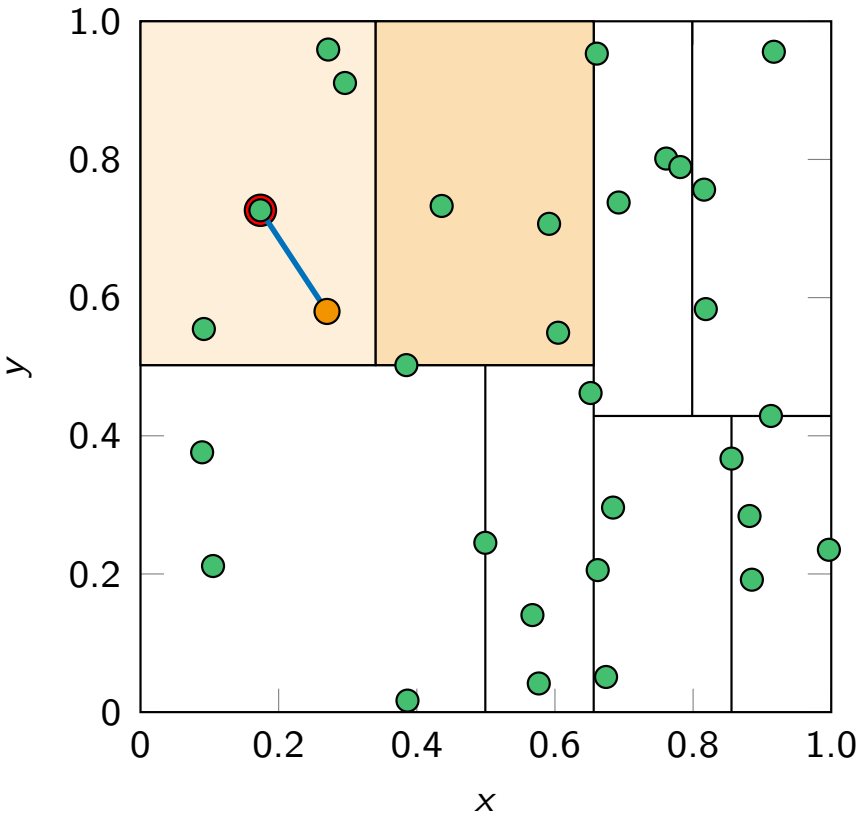
Nearest Neighbor Suche im k-d-Baum

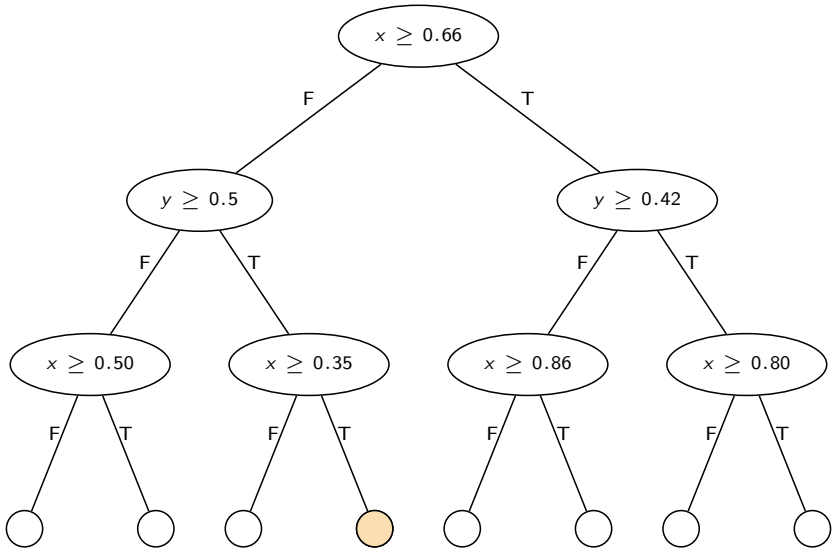




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

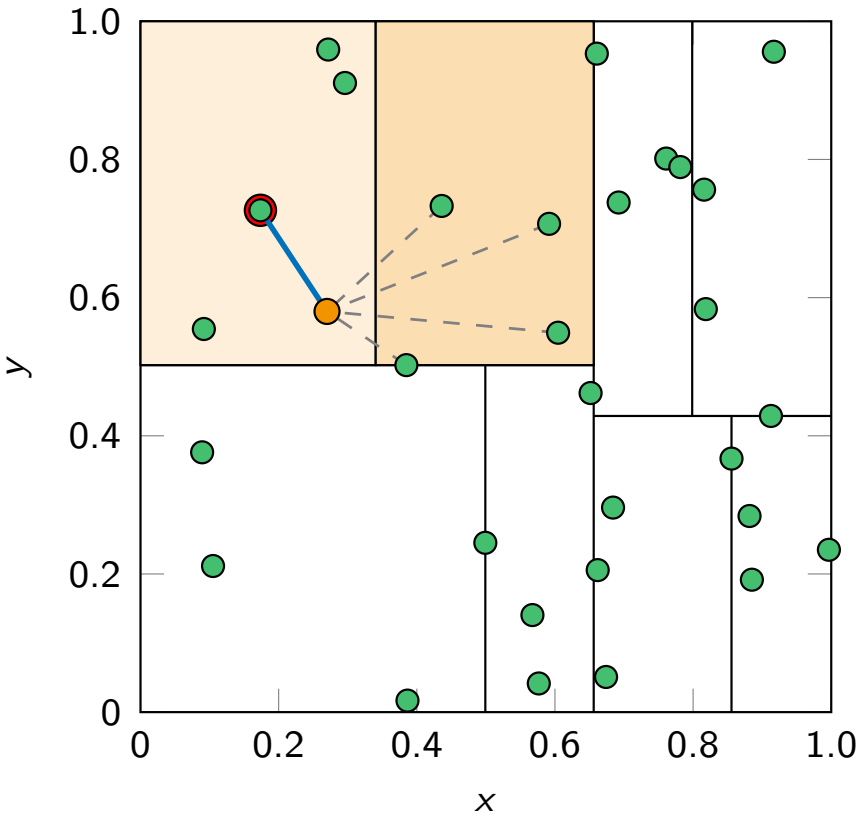
Nearest Neighbor Suche im k-d-Baum

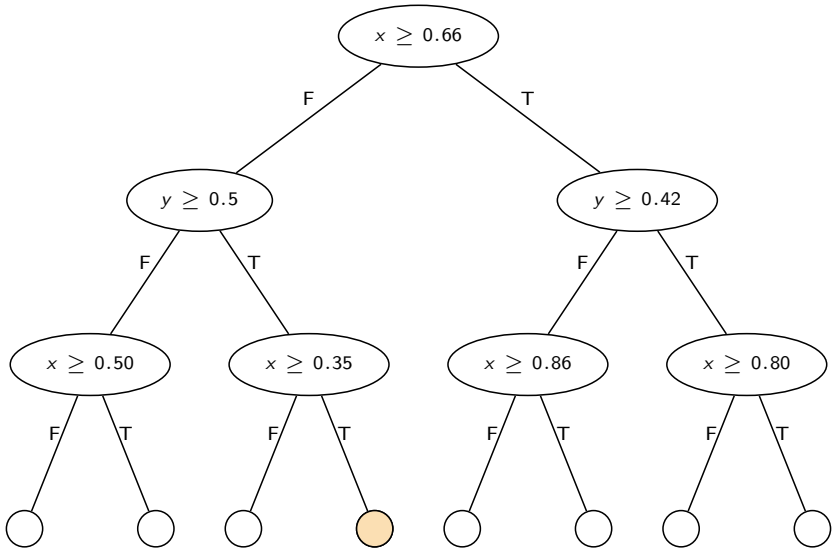




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

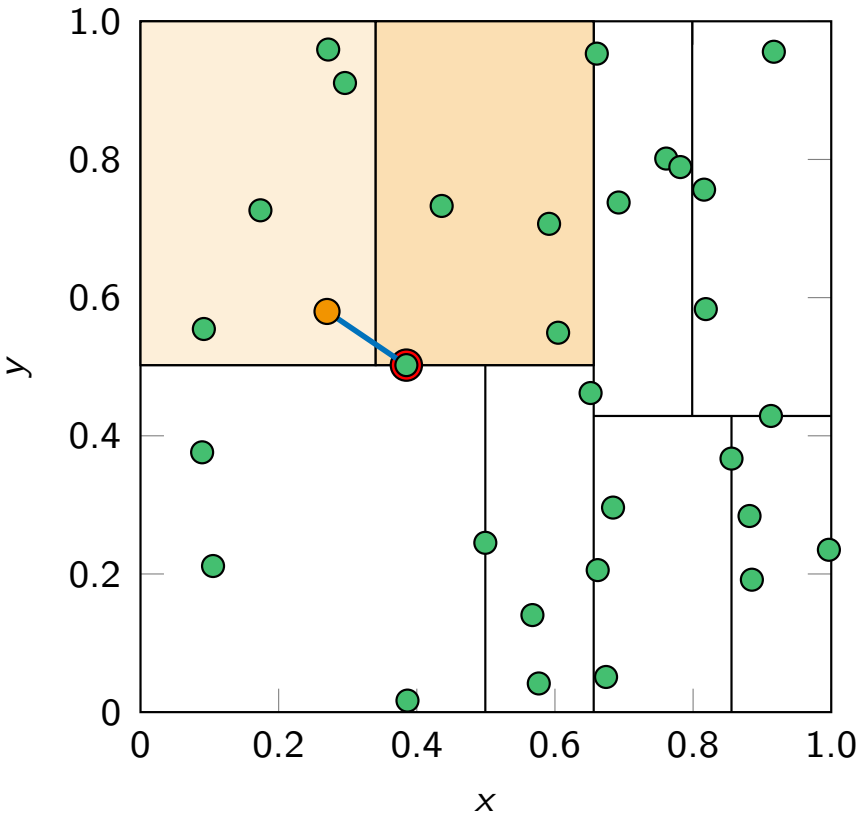
Nearest Neighbor Suche im k-d-Baum

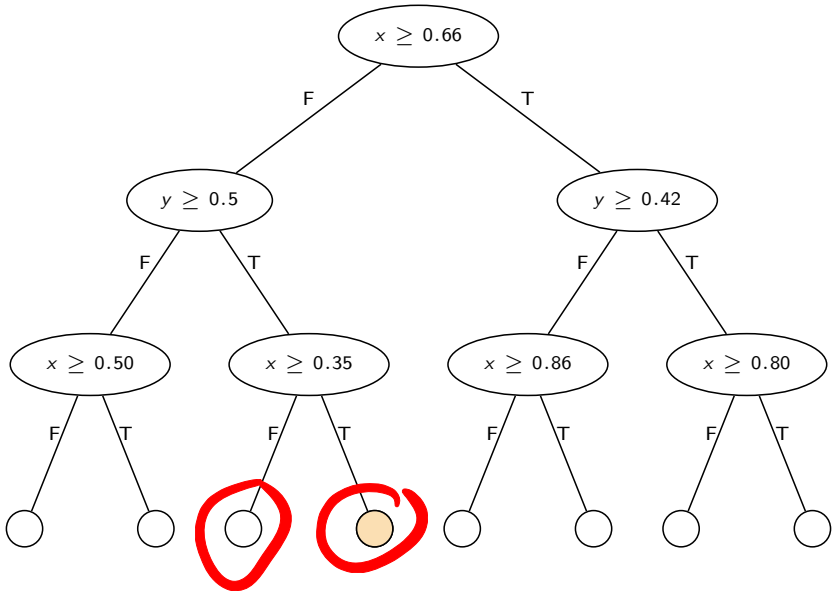




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

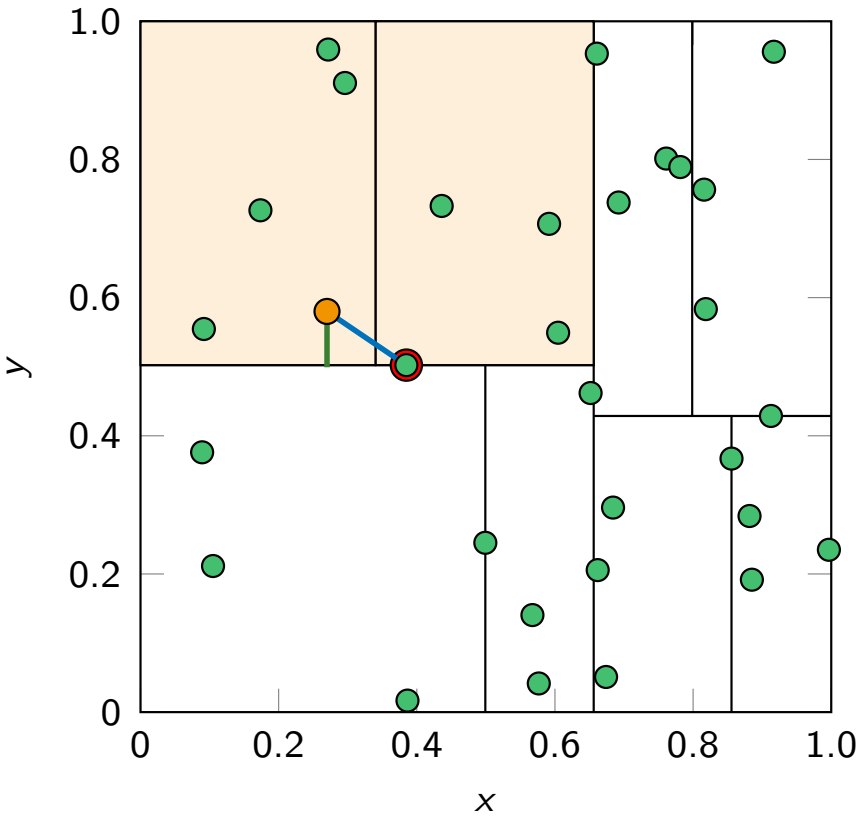
Nearest Neighbor Suche im k-d-Baum

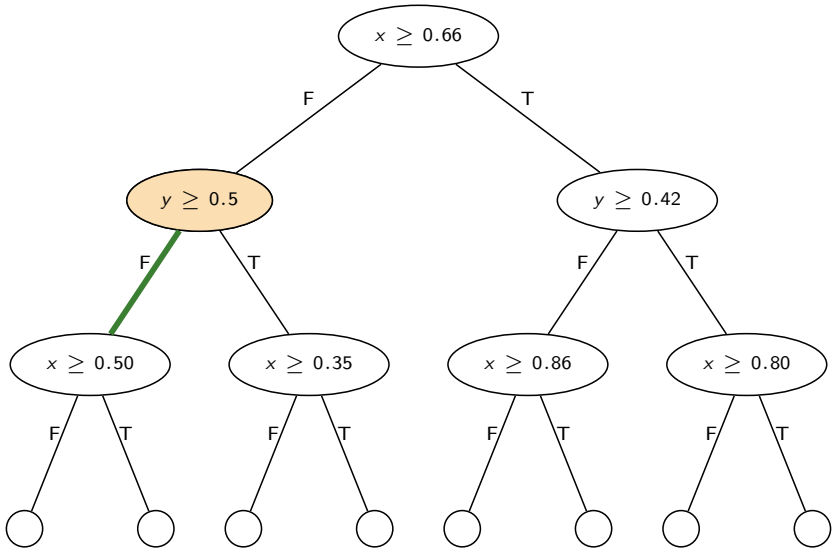




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

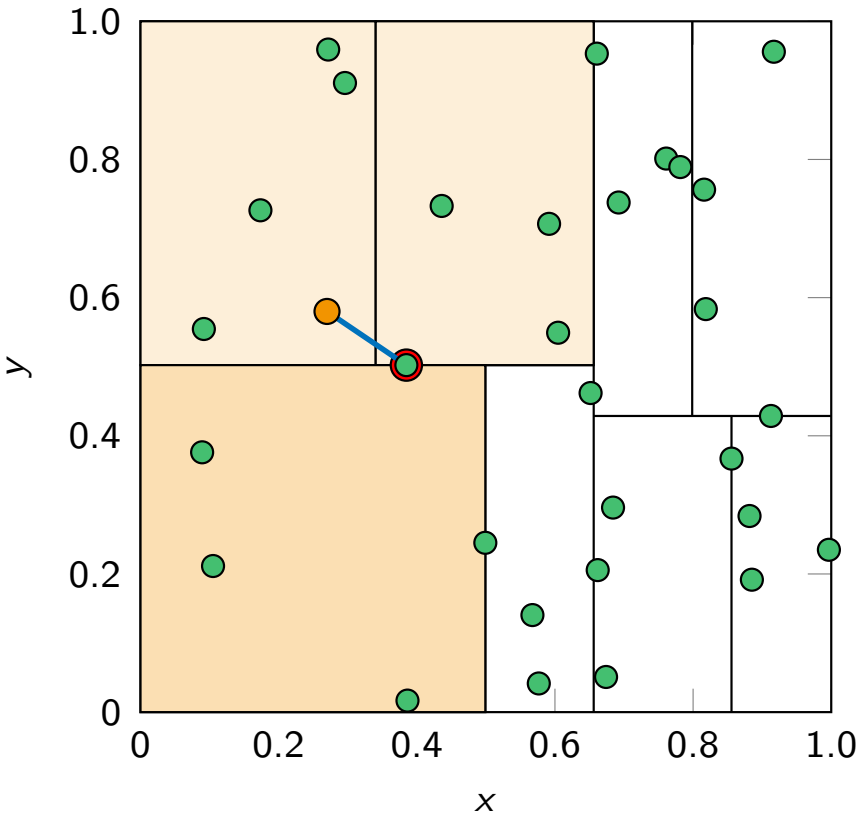
Nearest Neighbor Suche im k-d-Baum

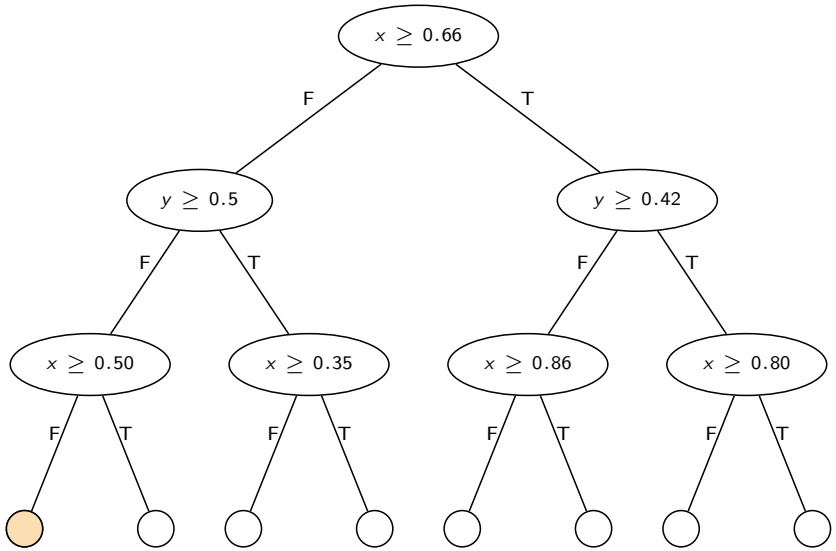




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

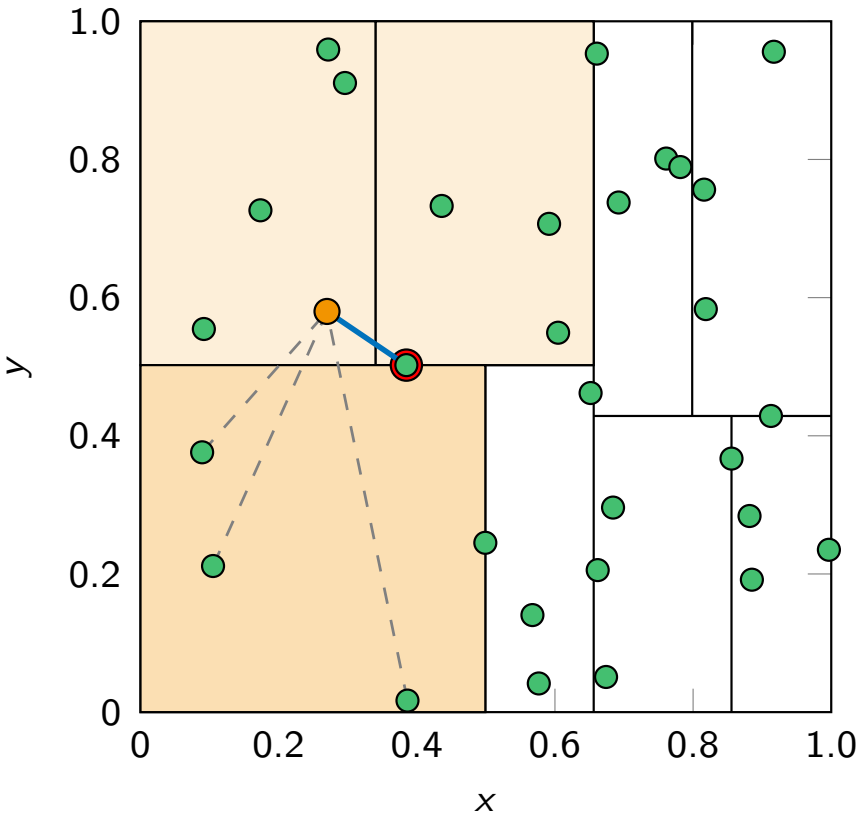
Nearest Neighbor Suche im k-d-Baum

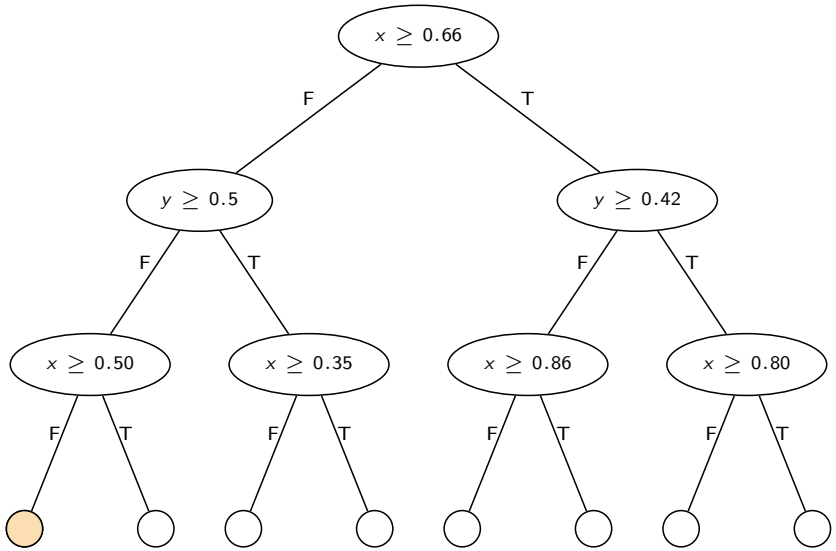




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

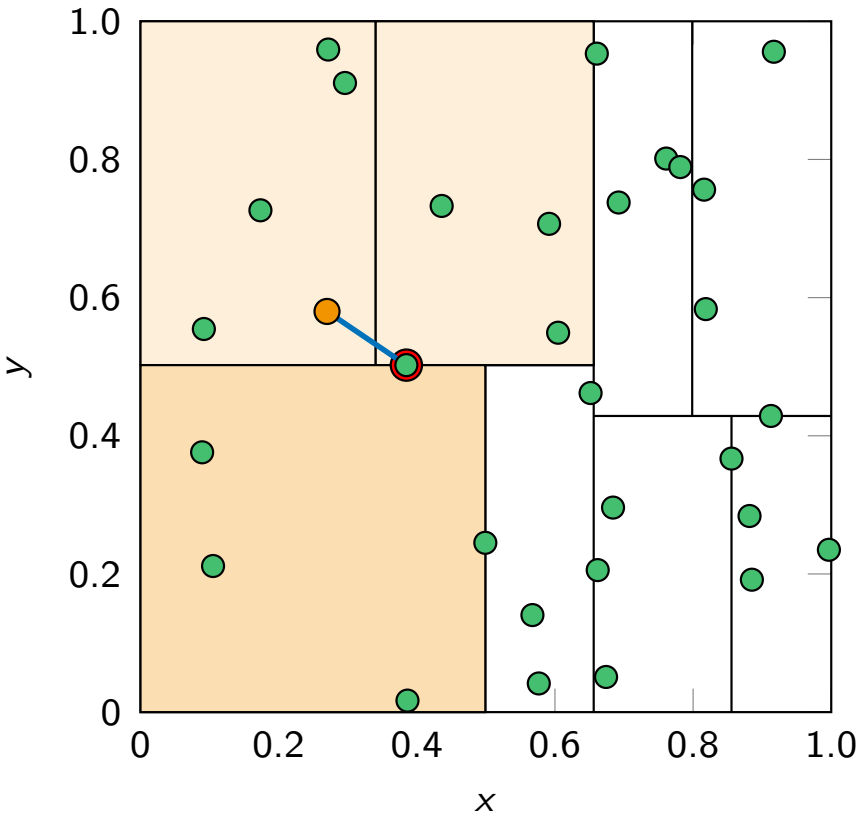
Nearest Neighbor Suche im k-d-Baum

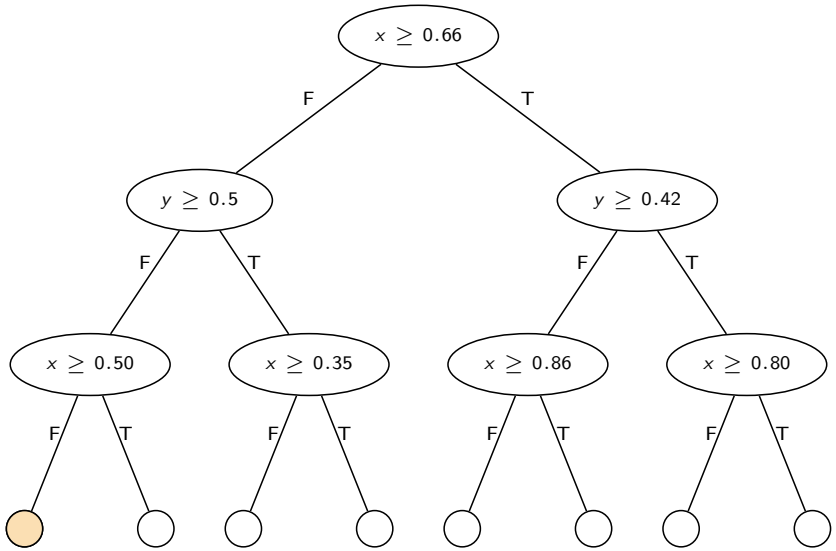




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

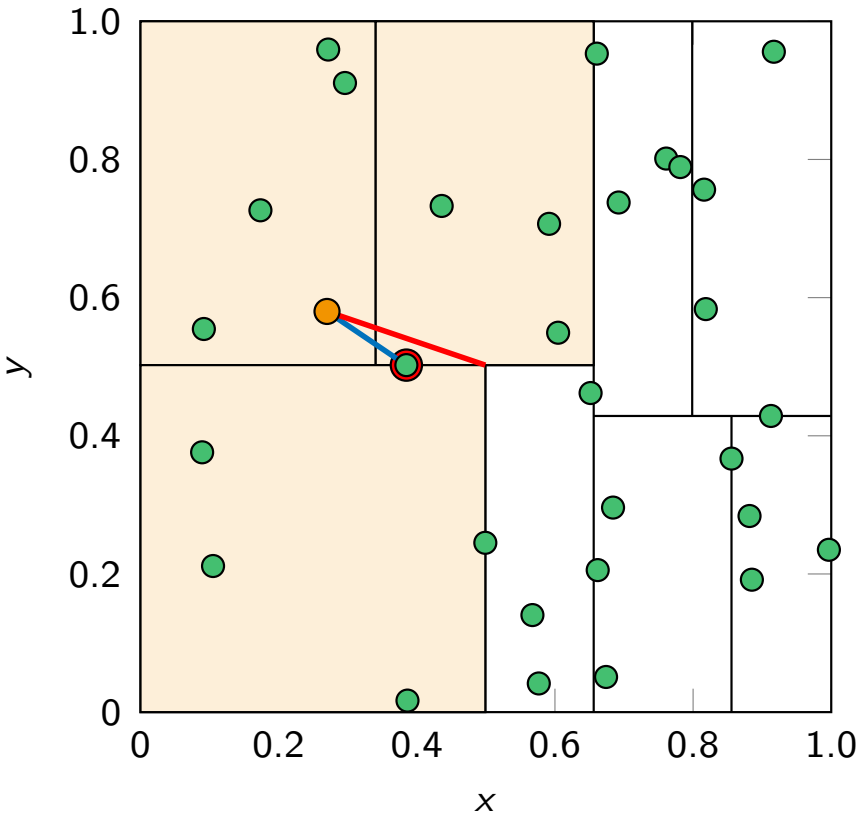
Nearest Neighbor Suche im k-d-Baum

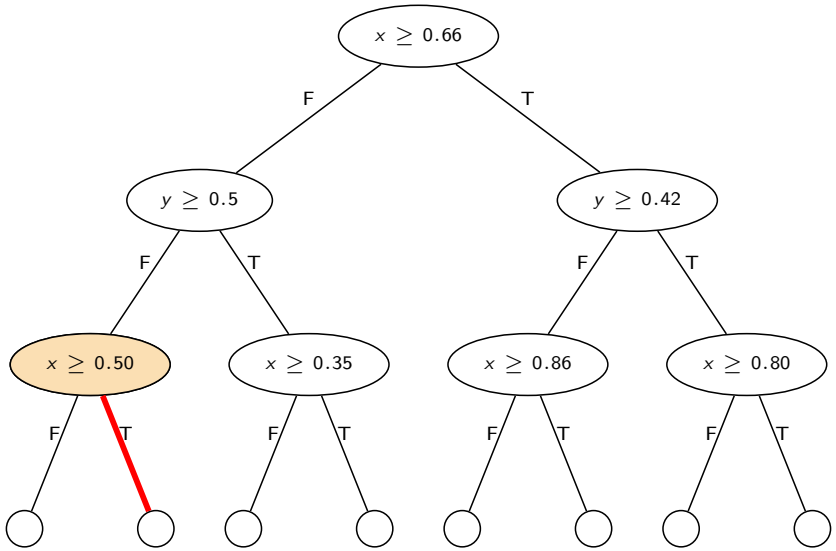




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

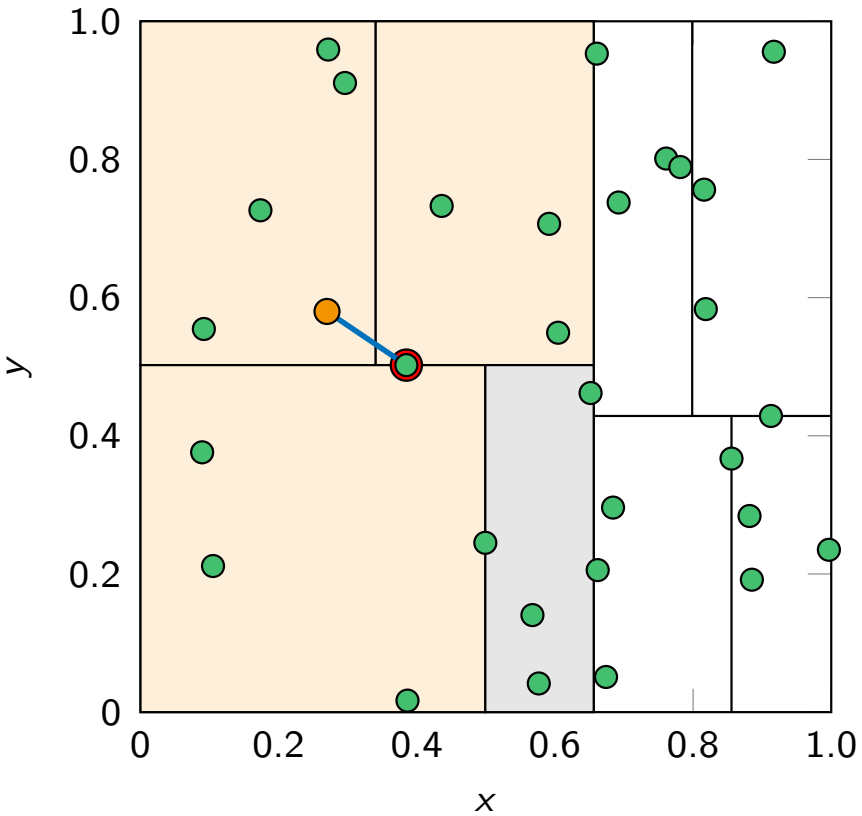
Nearest Neighbor Suche im k-d-Baum

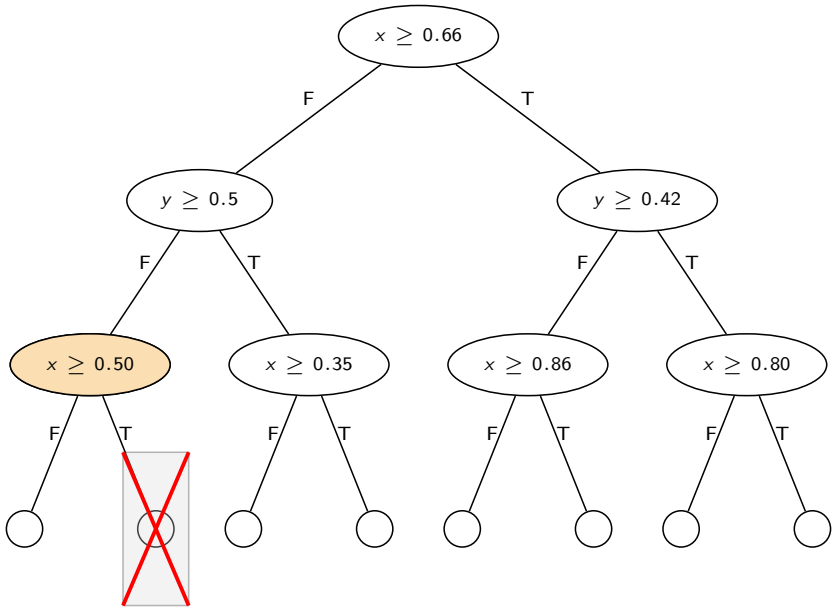




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

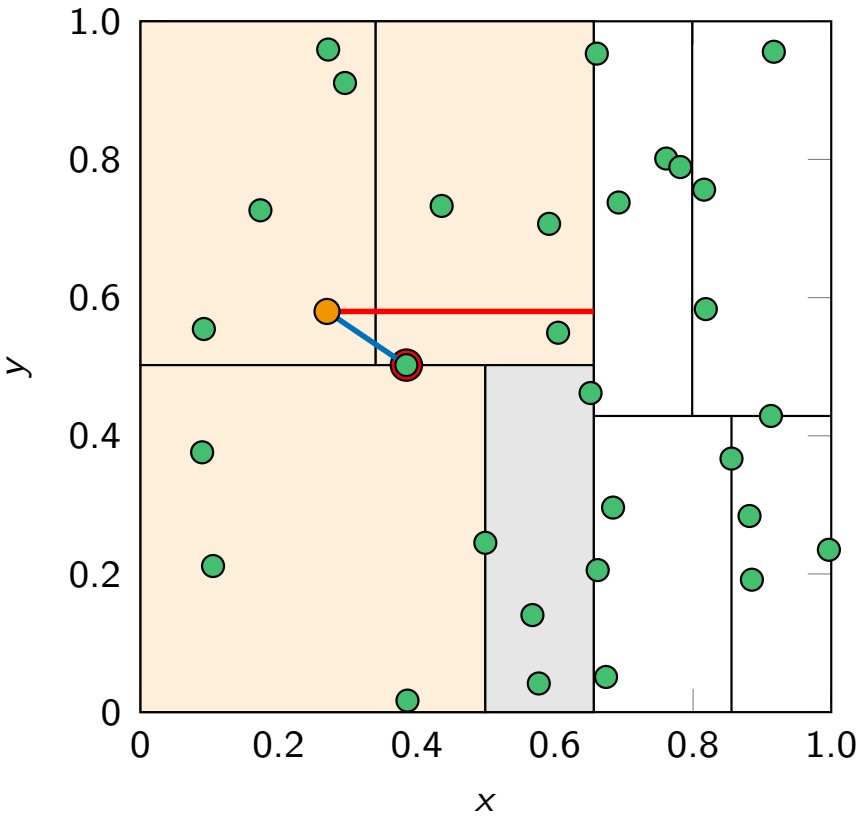
Nearest Neighbor Suche im k-d-Baum

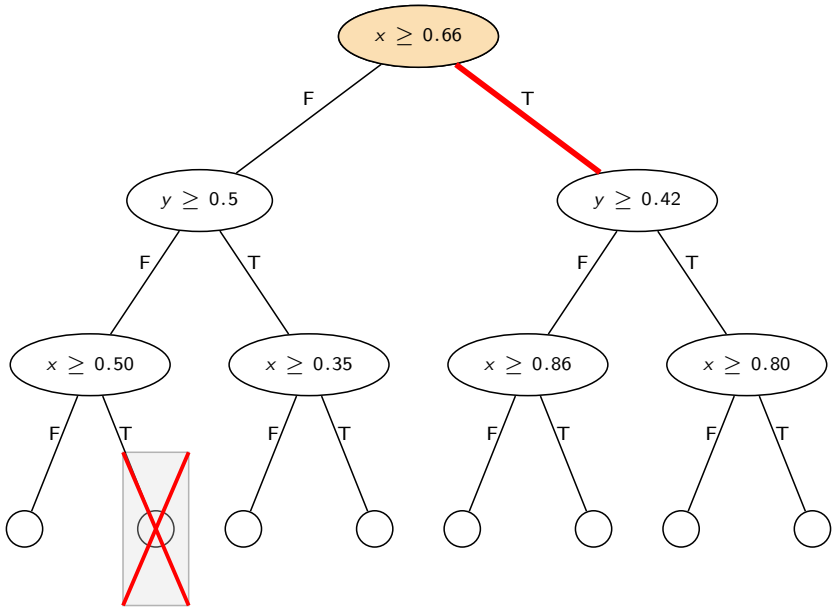




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

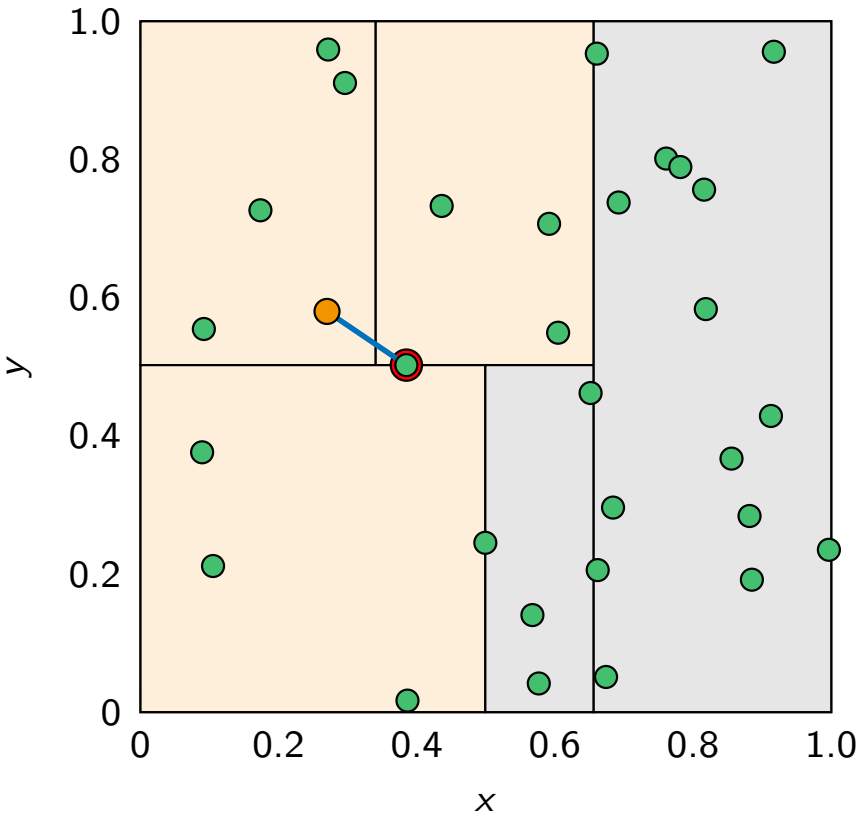
Nearest Neighbor Suche im k-d-Baum

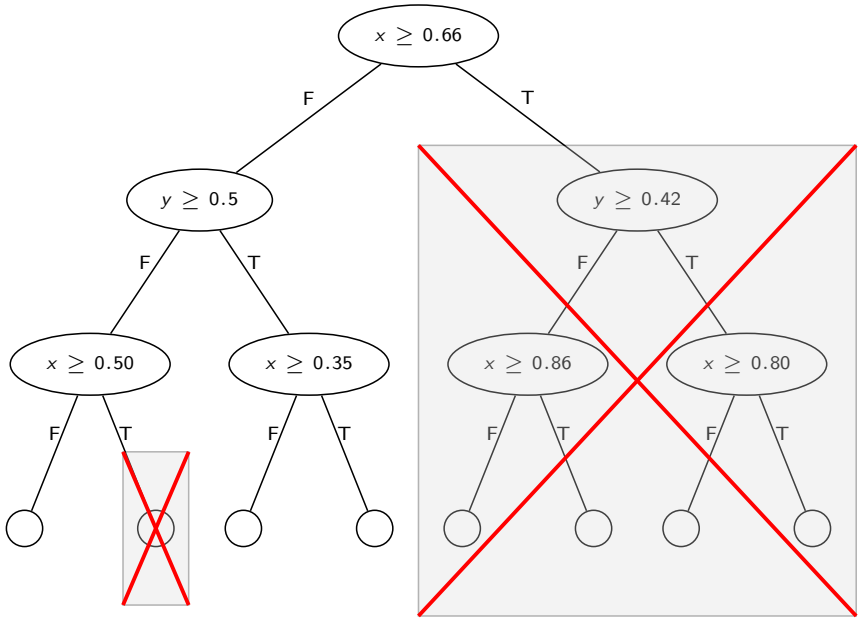




Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

Nearest Neighbor Suche im k-d-Baum





Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum: $\mathcal{O}(p \log(m))$ (bei m Datenpunkten und kleinem p).

Curse of Dimensionality

Nearest Neighbor-Suche im k-d-Baum

Durchschnittliche Komplexität für Nearest Neighbor-Suche in einem k-d Baum mit m Datenpunkten:

$$\mathcal{O}(\log(m)) \ .$$

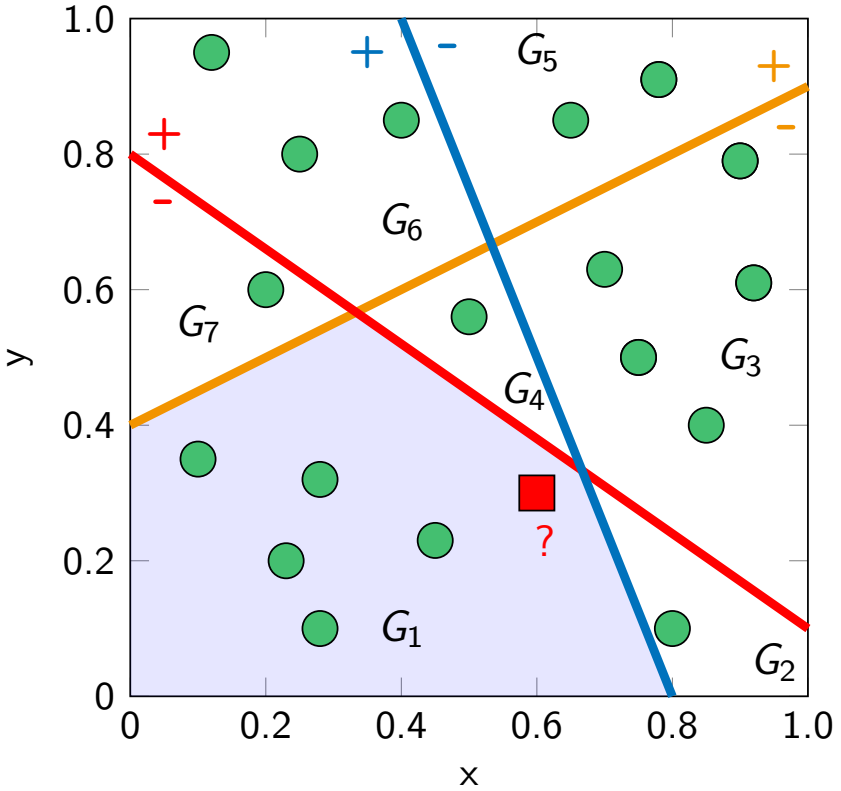
Worst Case Komplexität:

$$\mathcal{O}(m)$$

Im Fall von hochdimensionalen Daten tendiert die Nearest Neighbor Search bei einem k-d-Baum auf Grund des **Curse of Dimensionality** zur Komplexität $\mathcal{O}(m)$. In diesem Fall sollten andere Ansätze verfolgt werden (z.B. Locality-sensitive hasing, s. nächste Folie).

Locality-sensitive hashing

- Nearest Neighbor-Suche für hochdimensionale Probleme.
- Wähle zufällige Hyperebenen $h_1, \ldots, h_k$ mit $k \approx \log(m)$
- Raum wird durch diese in maximal 2^k Gebiete G_i aufgeteilt.
- Suche nach Nearest Neighbor nur im Gebiet des Query Points.
- Zeitkomplexität: $\mathcal{O}(kp + mp/2^k) \approx \mathcal{O}(p \log(m))$
- Nearest Neighbor kann in anderem Gebiet liegen, d.h. die Methode ist inexakt.
- Ansatz: wiederhole das Vorgehen mehrfach.



Distanzmaße für kontinuierliche Features

Die Auswahl des nächsten Nachbarn hängt von der verwendeten Metrik ab.

- **Euklidische Distanz** zwischen $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^p$:

$$\mathrm{dist}(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{k=1}^{p} (x_k - y_k)^2}$$

  Eigenschaften:
  - symmetrisch, behandelt alle Dimensionen „gleich“
  - sensitiv gegenüber extremen Ausreißern in einzelnen Attributen

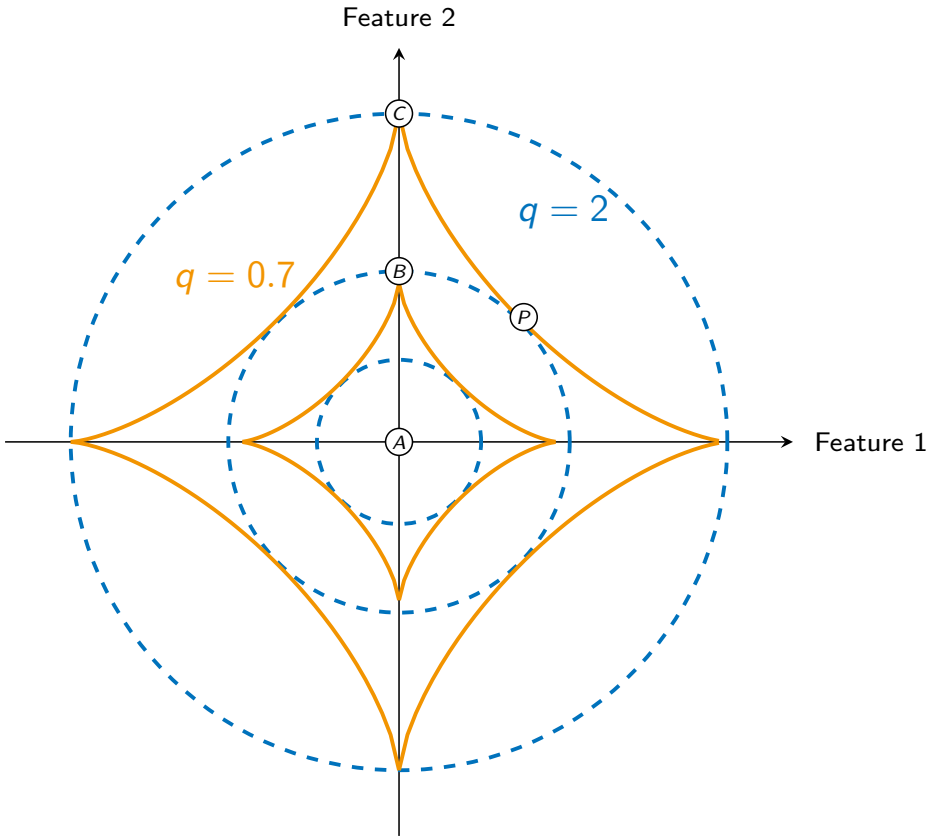
- **Minkowski-Distanz** (q-Norm) zwischen $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^p$:

$$\mathrm{dist}(\boldsymbol{x}, \boldsymbol{y}) = \left(\sum_{k=1}^{p} |x_k - y_k|^q \right)^{\frac{1}{q}}$$

  - $q = 2$: Euklidische Distanz
  - $q = 1$: Manhattan-Distanz
  - $q = 0$: Hamming-Distanz

Minkowski-Distanz

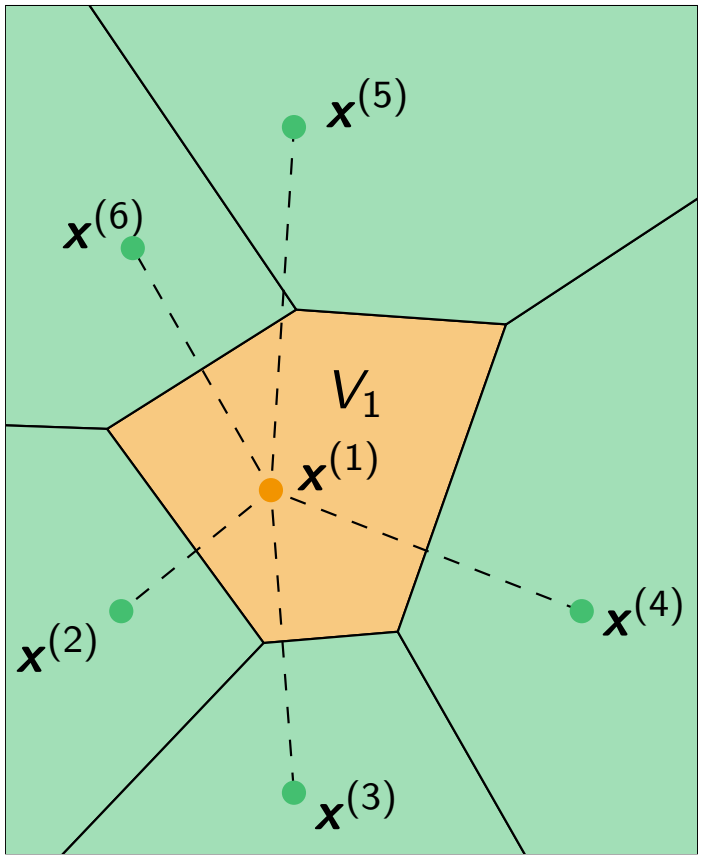
Visualisierung von Sphären für zwei verschiedene Werte von q:



Für $q = 0.7$ haben große Abweichungen, die nur in einer Komponente auftreten, vergleichsweise weniger Gewicht als für $q = 2$ (C wäre für $q = 2$ weiter von A entfernt als P, für $q = 0.7$ sind sie gleich weit von A entfernt).

Regionen der Klassenzuordnung bei Nearest Neighbor-Klassifikation

- Gegeben sei eine Menge von m Punkten $M = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ im p-dimensionalen Feature-Raum (hier: $p = 2$).
- Jeder Punkt $\boldsymbol{x}^{(i)}$ wird von einem Gebiet V_i umgeben, das alle Punkte enthält, die näher an $\boldsymbol{x}^{(i)}$ als an allen anderen Punkten $\boldsymbol{x}^{(j)} \in M$ sind.
- Jeder Punkt aus V_i wird durch die Nearest Neighbor-Klassifikation der gleichen Klasse wie $\boldsymbol{x}^{(i)}$ zugeordnet.
- Man nennt V_i eine **Voronoi-Zelle** zum Zentrum $\boldsymbol{x}^{(i)}$.



NN-Klassifikation - Voronoi-Diagramm

Ostbayerische Technische Hochschule Amberg-Weiden

Die Regionen der Klassenzuordnung bei der Nearest Neighbor-Klassifikation bilden ein **Voronoi-Diagramm** mit den Trainingsdaten $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}$ als Zentren.

- Jeder Datenpunkt $\boldsymbol{x}^{(i)}$ der Trainingsdatenmenge bildet das Zentrum einer Voronoi-Zelle, die alle Punkte des Feature-Raumes enthält, die zu diesem Punkt einen geringeren Abstand (gemessen meist in der euklidischen Metrik) haben als zu jedem anderen Datenpunkt $\boldsymbol{x}^{(j)}$, $j \neq i$, der Trainingsdatenmenge.
- Die Voronoi-Zellen bilden eine Partitionierung des gesamten Features-Raumes, d.h.

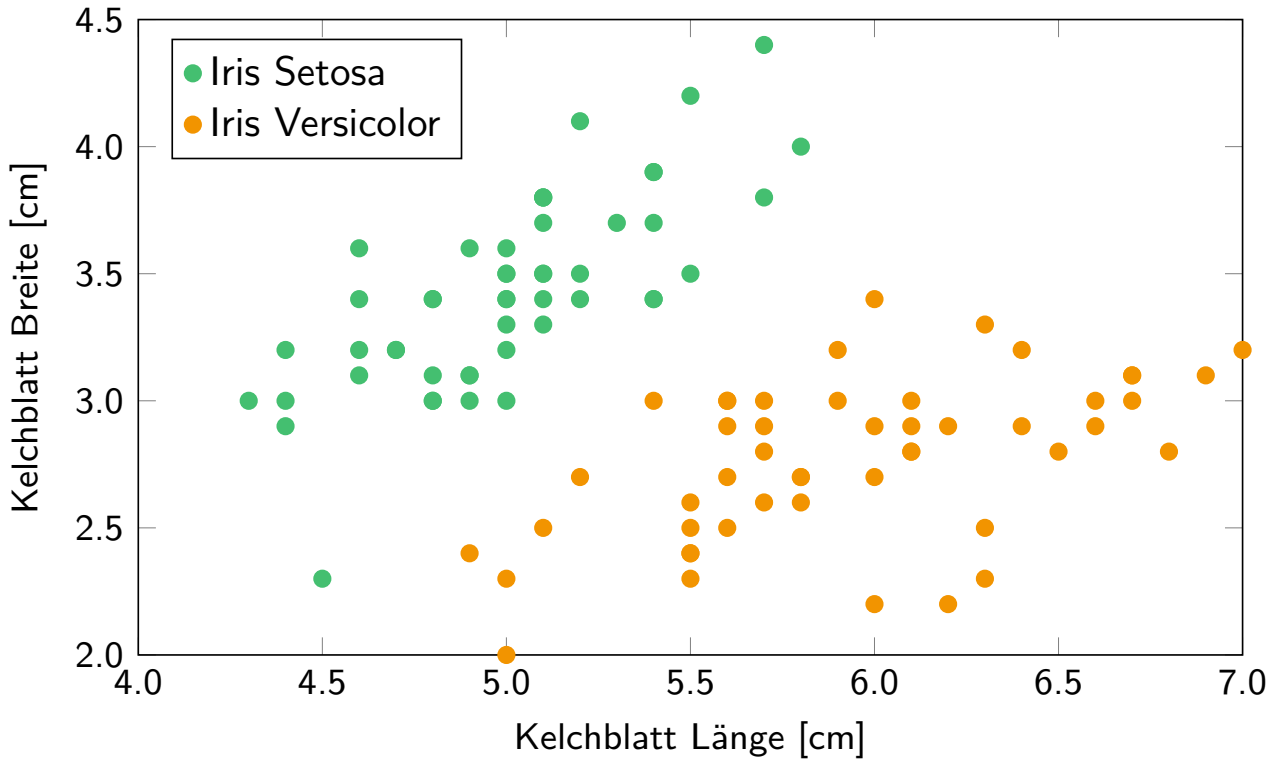
$$\mathbb{R}^p = \bigcup_{i=1}^{m} V_i \quad \text{mit} \qquad \operatorname{int}(V_i) \cap \operatorname{int}(V_j) = \emptyset \text{ für } i \neq j \ .$$

- Die Voronoi-Zellen sind (ggf. unbeschränkte) konvexe Polyeder (Schnitte von endlich vielen Halbräumen).

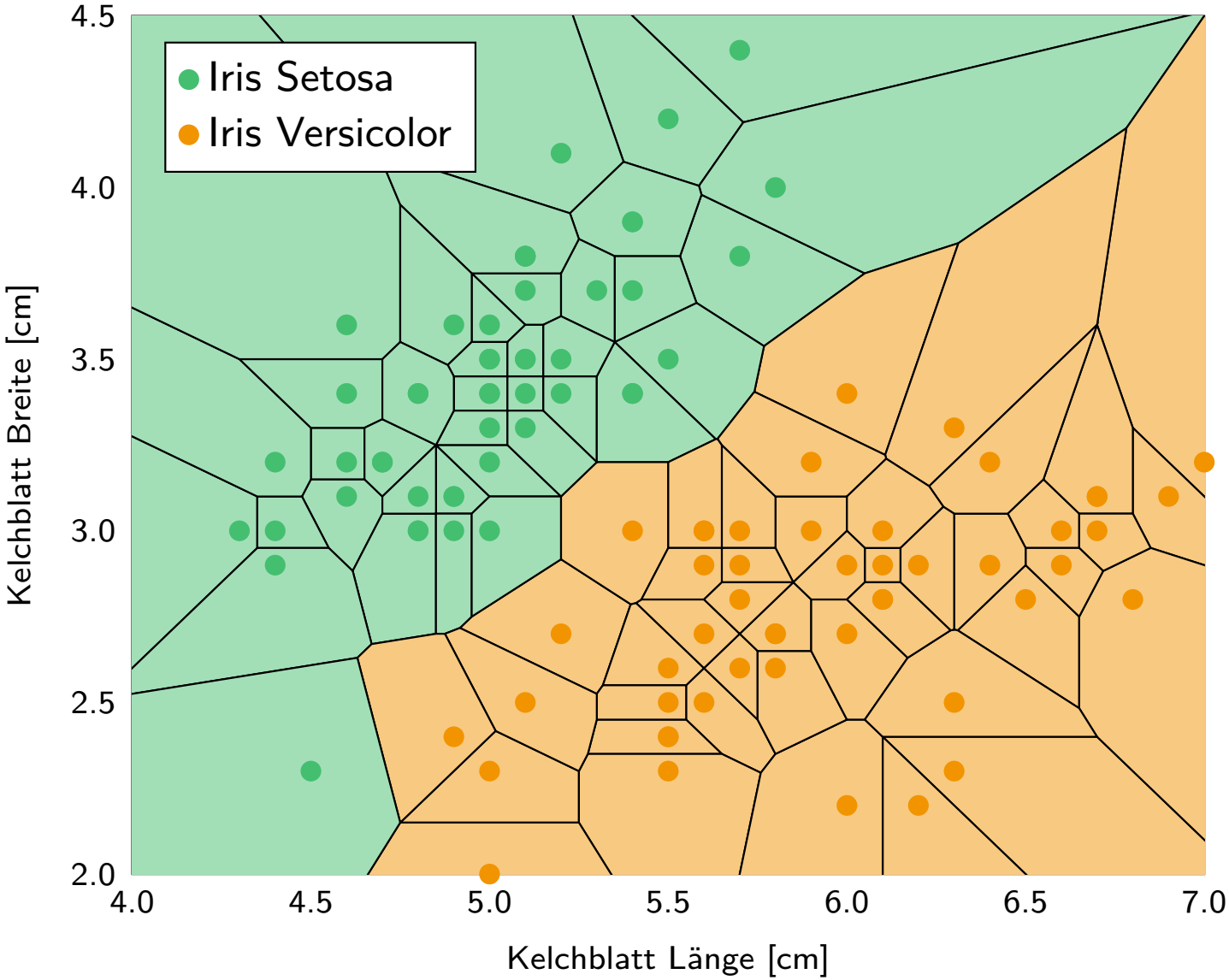
Nearest neighbor-Klassifikation für den Iris-Datensatz

Für das Schwertlilien-Beispiel betrachten wir zunächst nur die Klassen **Setosa** und **Versicolor** und wollen diese anhand der Features **sepal length** und **sepal width** klassifizieren.

In diesem Fall spannt der Feature-Raum eine Ebene auf und man kann die Samples wie folgt visualisieren:

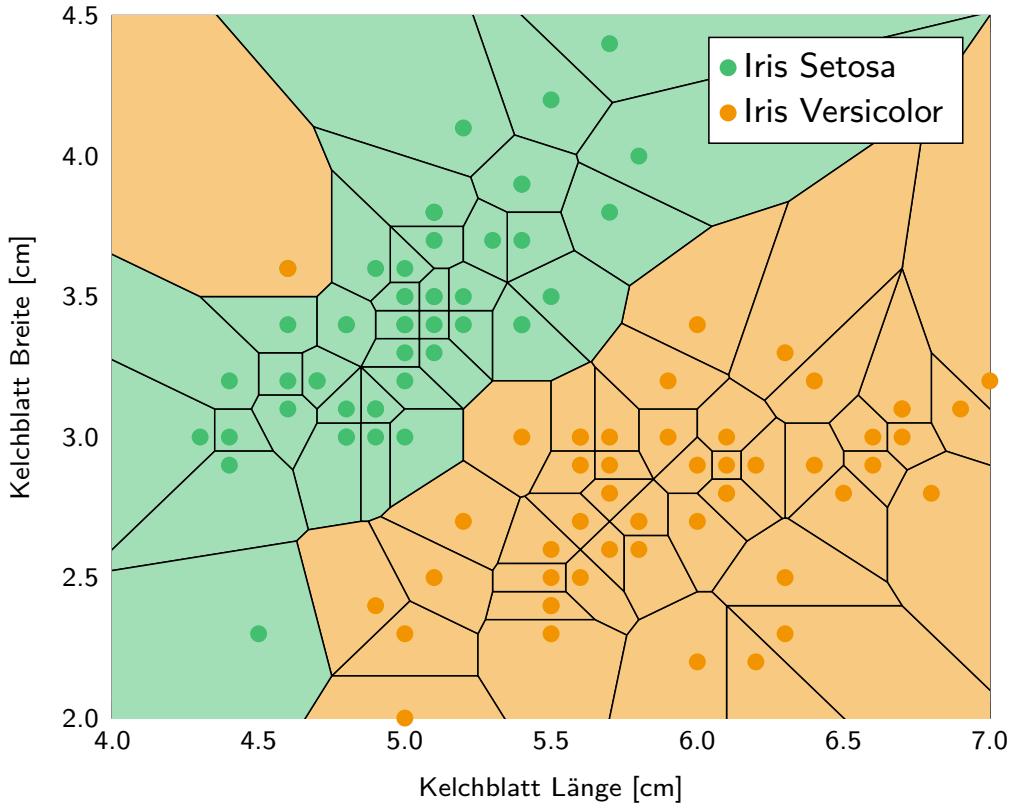


Voronoi-Diagramm am Beispiel des Iris-Datensatzes



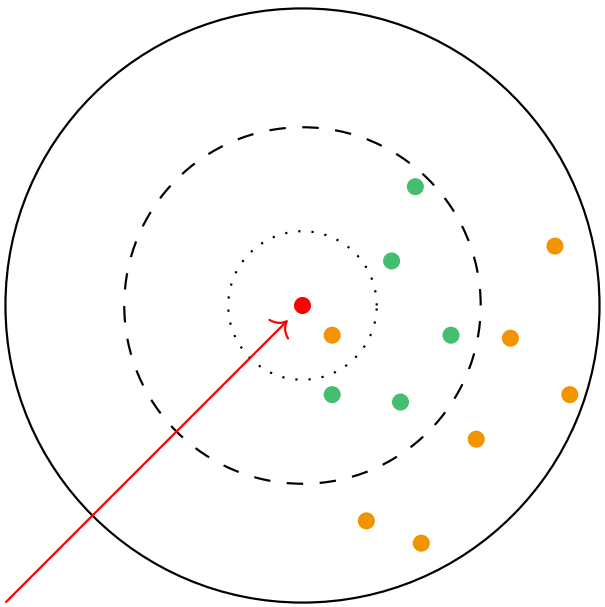
Nearest Neighbor-Klassifikation: Empfindlichkeit gegen Ausreißer

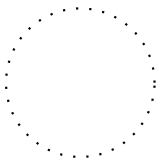
- Die Nearest Neighbor-Klassifikation ist empfindlich gegen Ausreißer.
- **Idee:** berücksichtige nicht nur den nächsten Nachbarn, sondern die k nächsten Nachbarn $\rightarrow$ k-Nearest Neighbor-Klassifikation (kNN)



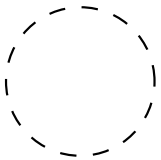
kNN - Grundidee

- Bestimme für den zu klassifizierenden Punkt die k nächsten Nachbarn und ordne die Klasse der Mehrheit dieser zu (ggf. gewichtet nach Distanz).
- Wird k zu klein gewählt, kann das Verfahren anfällig gegen Rauschen in den Daten werden.
- Wird k zu groß gewählt, befinden sich u.U. viele Objekte der falschen Klasse in der Entscheidungsmenge.

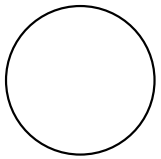




Entscheidungsmenge für $k = 1$



Entscheidungsmenge für $k = 6$



Entscheidungsmenge für $k = 12$

Hyperparameter

Das k-NN-Verfahren ist kein parametrisiertes Verfahren, hängt aber von sog. **Hyperparametern** ab, z.B.:

- Wert von k
- Distanzmaß
- Falls Minkowski-Distanz: Wahl von q
- Entscheidungsregel (z.B. einfache Mehrheitsentscheidung oder distanzgewichtete Entscheidung)
- Datenstruktur (z.B. k-d-Baum, Ball-Tree etc.)
- Anzahl an Datenpunkten pro Blatt in einem k-d-Baum

Im weiteren Verlauf der Vorlesung werden wir Techniken kennen lernen, um möglichst gute Werte für diese Hyperparameter zu ermitteln.

Zusammenfassung

- kNN ist ein einfach zu implementierendes und intuitives Verfahren zur Klassifikation
- Es ermöglicht einen nicht linearen Decision Boundary. Für kleines k tendiert das Verfahren zum Übertraining.
- kNN ist ein nicht-parametrisiertes, instanzbasiertes Verfahren des Supervised Learning.
- Es handelt sich um einen „lazy learner“.
- Das Verfahren hängt von verschiedenen Hyperparametern ab.
- Die Umsetzung sollte mit Hilfe geeigneter Datenstrukturen erfolgen.
- Der k-d-Baum ist nur für kleine Dimensionen geeignet.

Vor- und Nachteile

Ostbayerische Technische Hochschule Amberg-Weiden

Vorteile

- gute Intuition und Nachvollziehbarkeit
- einfach zu implementieren
- schnelle Anpassung an veränderte Trainingsdaten

Nachteile

- Hoher Speicherbedarf
- Klassifikation erfordert bisweilen die Verarbeitung des gesamten Trainingsdatensatzes
- liefert keine zusätzlichen Erkenntnisse über die Daten bzw. die zu Grunde liegenden Zusammenhänge
- Tendenz zu Übertraining