

Data Warehousing

Klassifikation

Ulf Leser

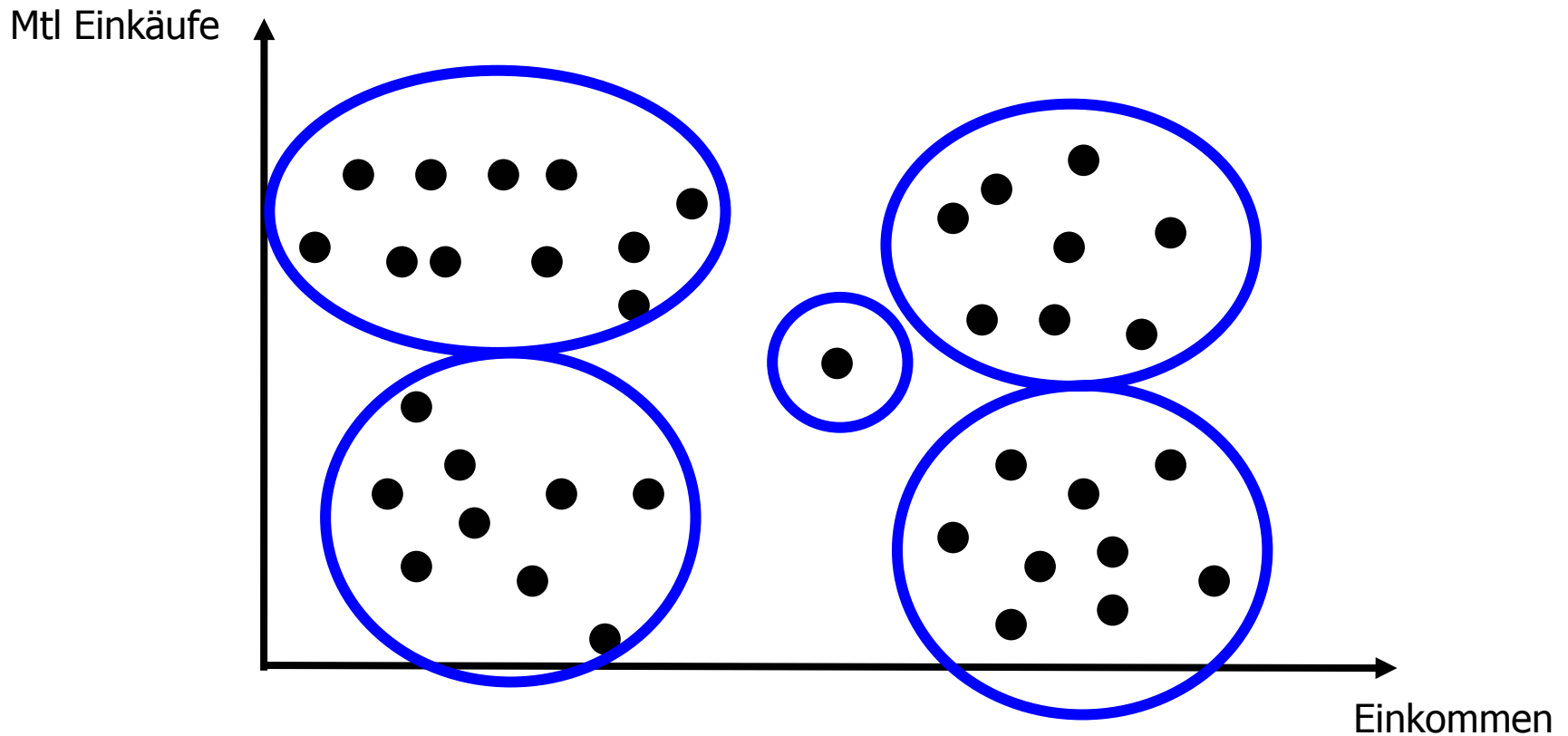
Wissensmanagement in der
Bioinformatik



Clustering

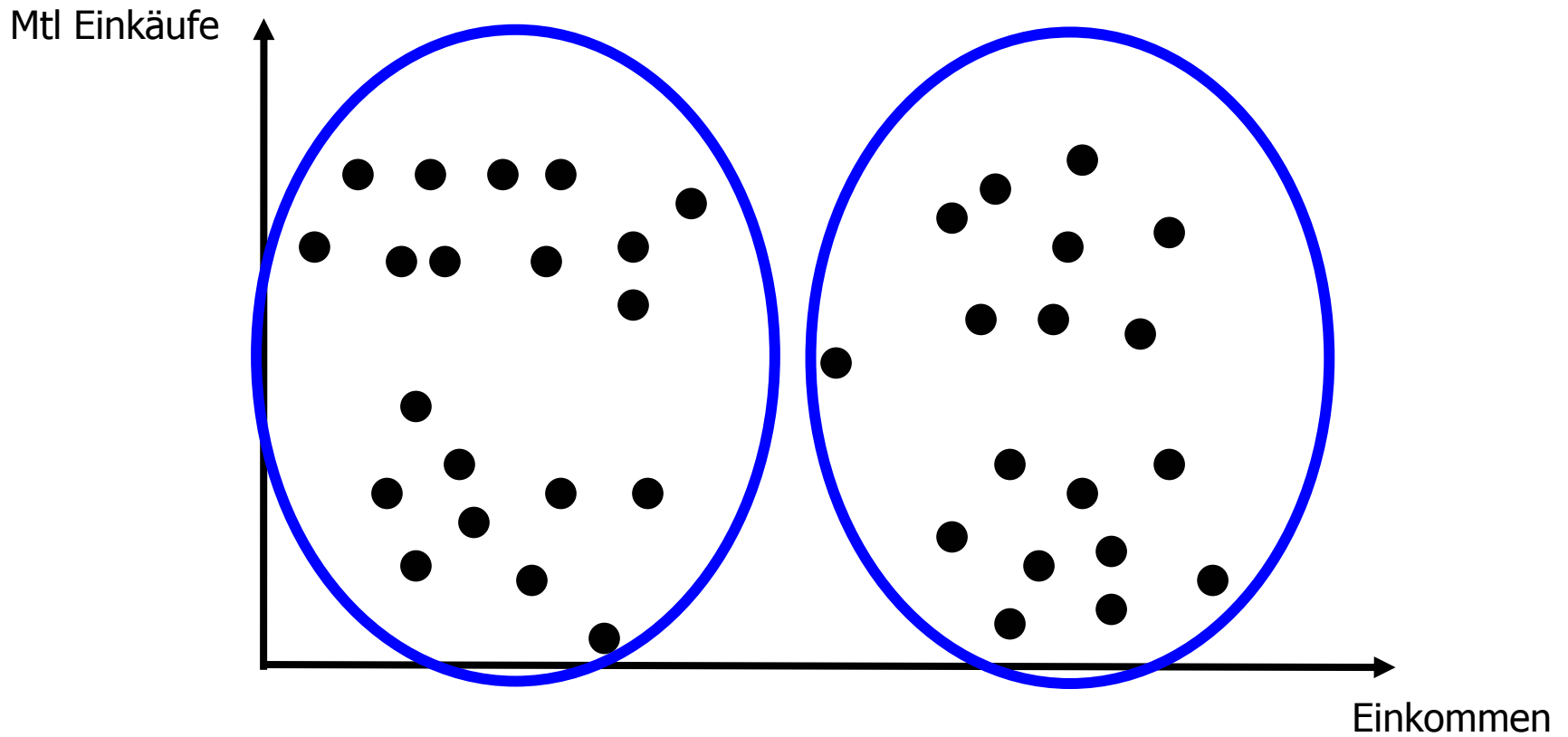
- Finde **Gruppen ähnlicher Objekte**
 - Ohne zu wissen wie viele Gruppen es geben soll
 - „**Unsupervised learning**“
 - (Mit der Gruppierung haben wir meist auch eine Klassifikation)
- **Anwendungen**
 - Segmentiere Kunden in Gruppen
 - Clustere Patienten in verschiedene Therapieanleitungen
 - Finde Typen von Sternen in astronomischen Karten
 - Welche Cluster von Genen reagieren ähnlich auf ein bestimmtes Medikament?
 - Welche Ergebnisse einer Websuche kommen aus dem selben Thema(encluster)
 - ...

Beispiel 1



- Vier Cluster und ein **Ausreißer**(-Cluster)
- Überlappungsfreie, konvexe Cluster

Beispiel 2



- Zwei Cluster
- Besser?

Güte eines Clusterings

- Intuitiv ist eine Gruppierung gut, wenn innerhalb **jedes Clusters alle Punkte nahe beieinander** liegen
- Definition
*Sei f eine Klassenzuordnung für Objekte O zu einer Menge von Klassen C mit $|C|=k$. Sei m_c der Mittelpunkt aller Objekte der Klasse $c \in C$, und sei $d(o, o')$ der Abstand zwischen zwei beliebigen Punkten. Dann ist die **k -Güte von f** :*

$$q_k(f) = \sum_{c \in C} \sum_{f(o)=c} d(o, m_c)$$

- Bemerkung
 - Zur Bestimmung von Mittelpunkten kommen wir gleich
 - Auch die **Einschränkung auf k -Güte** erklärt sich gleich

Clustergüte zweiter Versuch

- Definition

*Sei $f: O \rightarrow C$ mit $|C|$ beliebig. Sei $\text{dist}(o, C_i)$ der **mittlere Abstand** von o zu allen Punkten eines Clusters C_i .*

Damit definieren wir

- $a(o) = \text{dist}(o, f(o))$

- $b(o) = \min(\text{dist}(o, C_i))$ mit $C_i \neq f(o)$

- Die **Silhouette eines Punktes** $s(o)$ ist
$$s(o) = \frac{b(o) - a(o)}{\max(a(o), b(o))}$$

- Die **Silhouette von f** ist $\sum s(o)$

- Bemerkung

- $b(o)$: Verschlechterung, wenn es Cluster $f(o)$ nicht gäbe (und o dem nächsten Cluster zugeordnet werden würde)

- Das heißt

- $s(o) \approx 0$: Punkt liegt zwischen zwei Clustern

- $s(o) \rightarrow 1$: Punkt liegt näher am eigenen als am nächsten Cluster

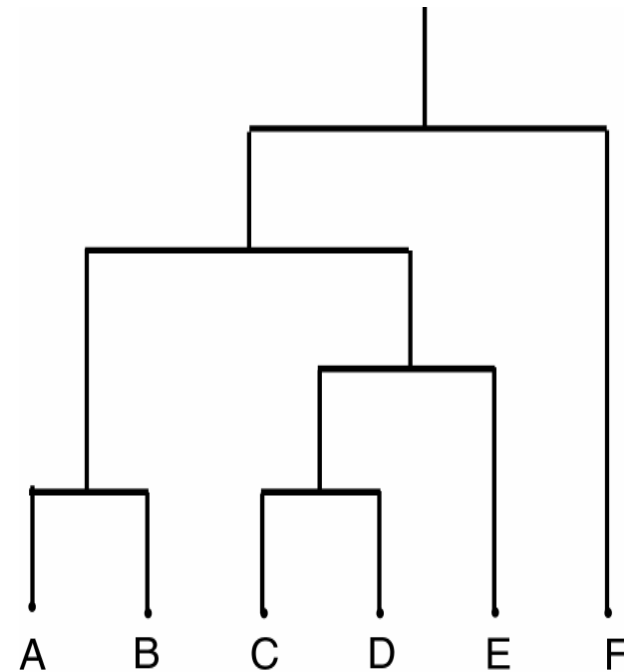
- $s(o) \rightarrow -1$: Punkt liegt näher am nächsten Cluster als am eigenen

Clusteralgorithmen

- **Hierarchische Verfahren:** hierarchisch geschachtelte Cluster
 - Bottom-Up: Ausgehend von $|C|=|O|$ werden sukzessive Cluster verschmolzen bis nur noch ein Cluster da ist
 - Top-Down: Ausgehend von $C_0=O$ werden sukzessive Cluster geteilt bis jedes Objekt sein eigener Cluster ist
 - Benötigen kein k , aber sind eher teuer
 - Berechnen nicht unmittelbar Cluster (weiterer Schritt notwendig)
- **Partitionierende Verfahren:** Zerlegung der Punktmenge in k Cluster
 - Benötigen die Anzahl k der Cluster als Parameter
- **Dichte-basierte Verfahren**
 - Suchen nach Subräumen mit hoher Punktdichte und fassen die zu Clustern zusammen
 - Tendenziell langsam
 - Können sehr gut durch multidimensionale Indexstrukturen unterstützt werden

Hierarchisches Clustering

- Auch: UPGMA
 - Unweighted Pair-group method with arithmetic mean
- Bottom-Up Berechnung eines binären Baums (**Dendogramm**)
- Algorithmus
 - Berechne **Abstandsmatrix M**
 - Ähnlichkeit aller Objekte zu allen anderen Objekten
 - Wähle Paar o_1, o_2 mit dem kleinsten Abstand
 - Berechne $x = m(o_1, o_2)$ (den Mittelpunkt)
 - Lösche o_1, o_2 aus M; füge x ein
 - Berechne Abstand von x zu allen verbleibenden Objekten/Clustern in M
 - Als Mittelwert der Abstände von o_1 und o_2
 - Iteriere, bis M leer ist



Eigenschaften

- Vorteile
 - Konzeptionell **einfach**, hübsche, irgendwie intuitive Grafiken
 - Keine Festlegung auf Anzahl Cluster notwendig
- Nachteile
 - Benötigt die **Abstandsmatrix** als Eingabe
 - $|O|=n$: $O(n^2)$ Platz und $O(n^2)$ Zeit
 - Plus Zeit zur Berechnung der Cluster
 - Berechnet keine Cluster
 - Weiterer Schritt (wie?) notwendig
- Fazit
 - Kaum anwendbar für viele (>10000) Objekte
 - Eher ein Umweg als Clustering

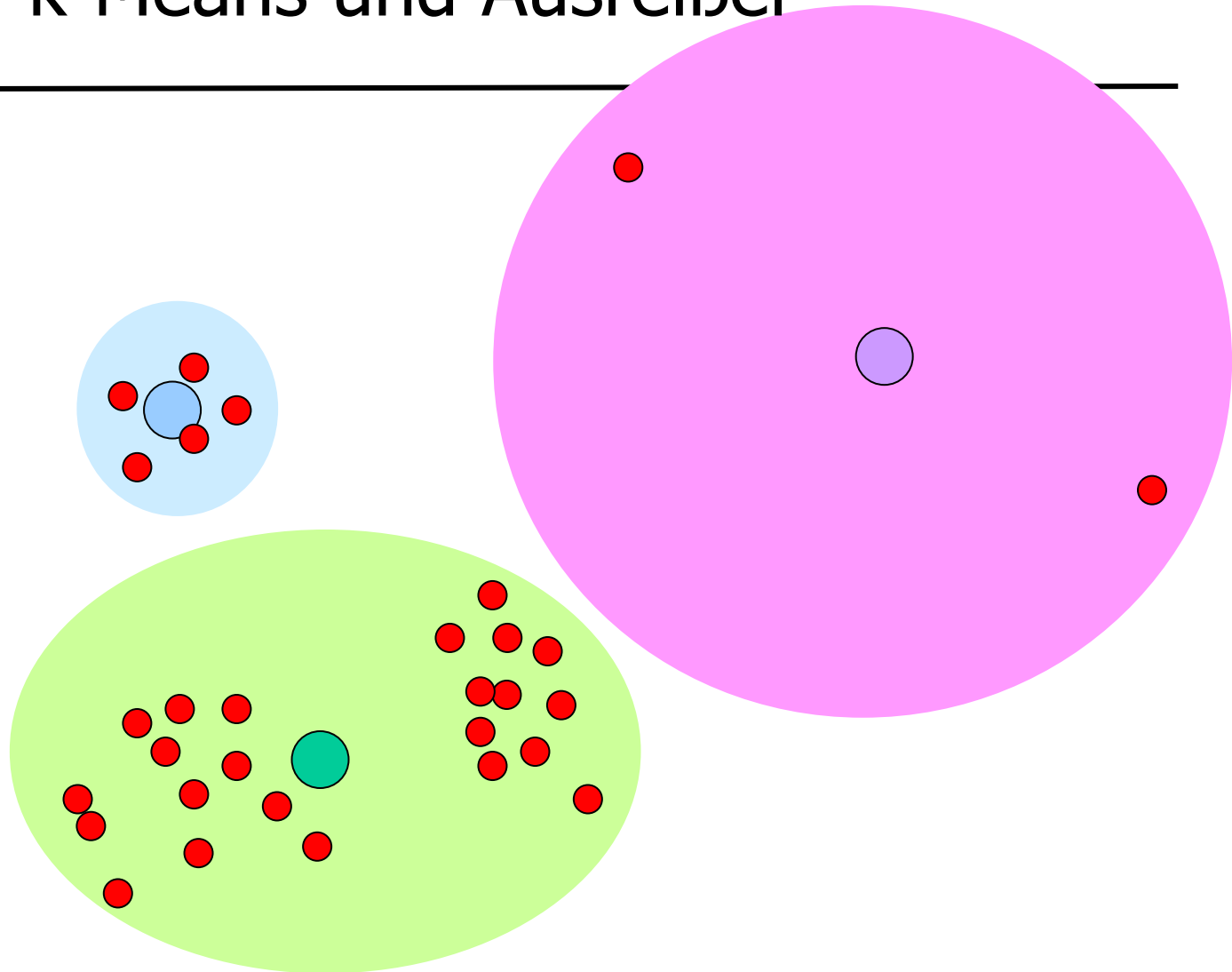
K-Means

- K-Means ist wahrscheinlich die bekannteste Clusteringmethode
 - Mit sehr vielen Varianten
- Benötigt die **Anzahl k von Clustern als Parameter**
- Berechnet ein lokales Optimum bezüglich k -Güte
 - Keine Garantien für globale Optimalität bzgl. k -Güte
- Algorithmus
 - Wähle zufällig k Clustermittelpunkte
 - Iteriere
 - Für alle Objekte
 - Berechne Abstand jedes Objekts zu jedem Clustermittelpunkt
 - Weise Objekt seinem **nächsten Clustermittelpunkt** zu
 - Wenn sich keine Objektzuordnung mehr geändert hat, dann STOP
 - Für alle Cluster
 - Berechne **neues Clusterzentrum**

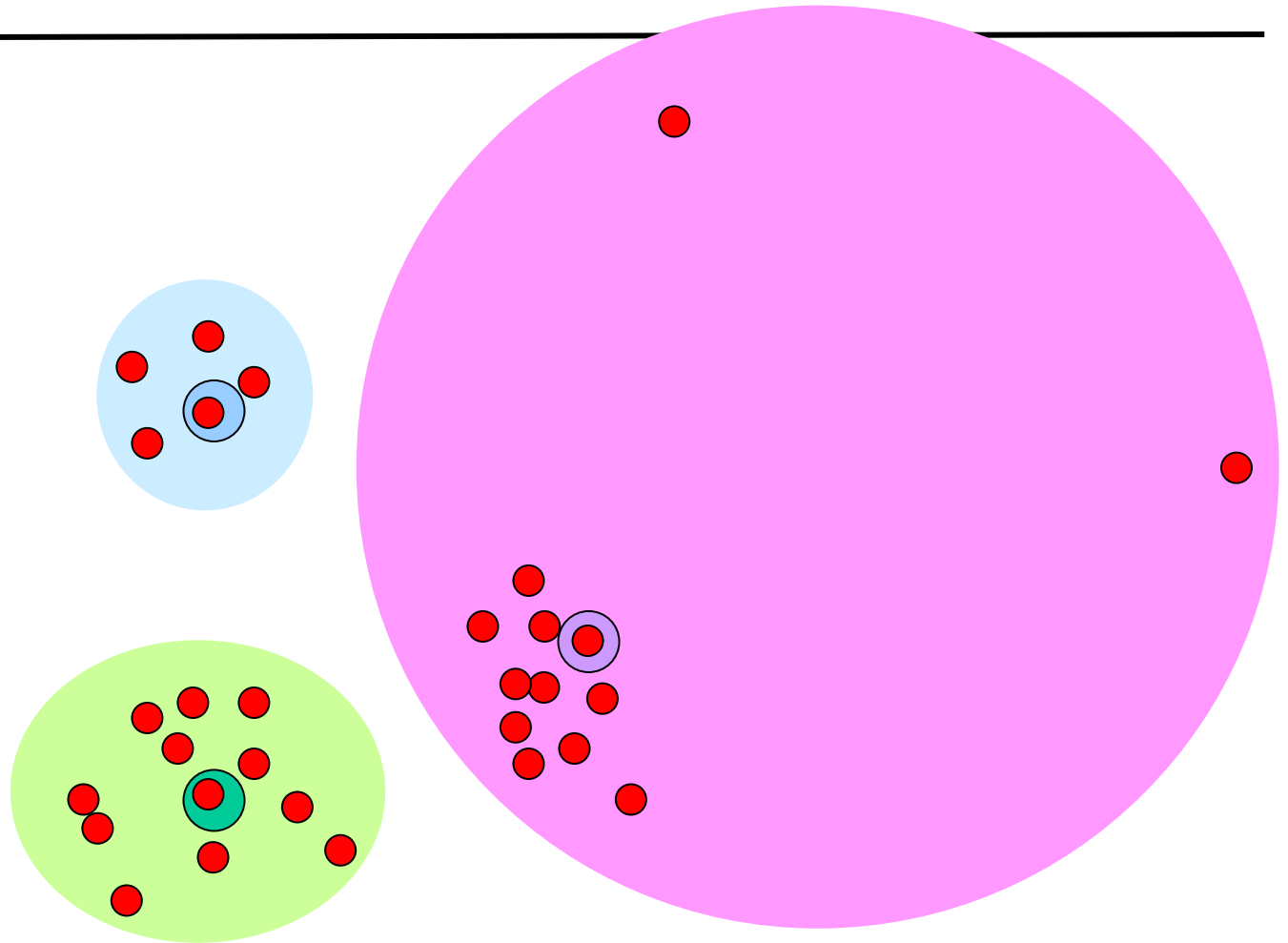
Eigenschaften

- In der Praxis konvergiert der Algorithmus relativ schnell (5-10 Läufe)
- Wenn l die Zahl der Durchläufe ist, brauchen wir
 - l Iterationen
 - Neuzuordnung: $n \cdot k$ Vergleiche Objekte-Zentren
 - Clusterbestimmung: n Vektoradditionen, verteilt auf k Cluster
 - Zusammen: $O(n \cdot k \cdot l)$
 - Das ist schnell
 - Insbesondere benötigen wir **keine Distanzmatrix**
- Nachteil: Welches k nehmen wir?
 - Alternative: Verschiedene k probieren
 - Silhouette zur Güteabschätzung verwenden
- Sehr empfindlich gegenüber **Ausreißern**
 - Verzerren die Struktur der Cluster
 - Es werden immer alle Punkte geclustered

k-Means und Ausreißer

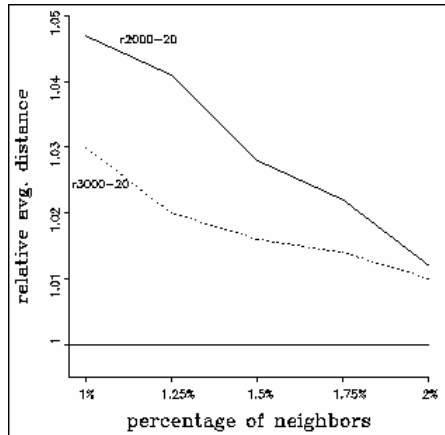


k-Medoid und Ausreißer



Vergleich [ES00]

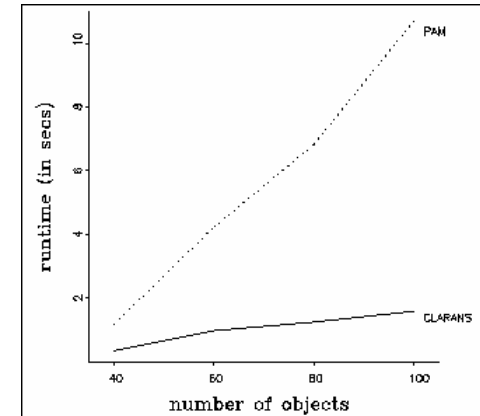
Qualität



TD(CLARANS)

TD(PAM)

Laufzeit



Quelle:
[ES00]

- Unwesentlich schlechtere Ergebnisse (1-5%)
- Viel bessere Laufzeit (nahezu linear)
- Nicht untypisch: Wenn die Daten „gut“ clustern, dann findet man diese Cluster sehr schnell
 - Zuordnung der wenigen problematischen Objekte benötigt dann viel Zeit, bringt aber nur wenig Verbesserung

Dichtebasiertes Clustering [EKSX96]

- Sucht nach **Regionen hoher Dichte**
 - Anzahl Cluster ist nicht vorbestimmt
 - Findet auch **nicht konvexe** Cluster
- Bekanntester Vertreter: **DBSCAN**
- Wie definiert man „dichte“ Bereiche?
 - Jeder Punkt eines Clusters hat **viele nahe Nachbarn**
 - Alle Punkte eines Clusters sind über nahe Nachbarn voneinander erreichbar

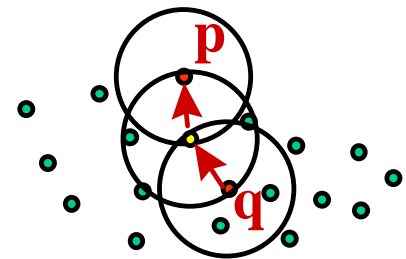
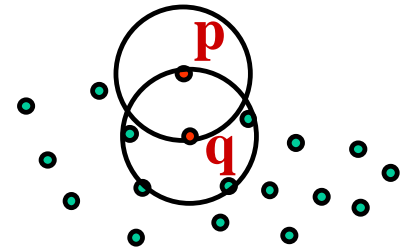


Grundbegriffe

- Definition

Geg. Parameter ε („Nachbar“) und minpts („viele“). Sei $N_\varepsilon(o)$ die ε -Nachbarschaft von Punkt o .

- Ein Objekt o heißt **Kernobjekt**, wenn $|N_\varepsilon(o)| \geq \text{minpts}$
- Ein Objekt p ist **direkt dichte-erreichbar** von einem Objekt q , wenn q ein Kernobjekt ist und $p \in N_\varepsilon(q)$
 - p muss kein Kernobjekt sein (Rand)
- p ist **dichte-erreichbar** von q , wenn es eine Kette von direkt dichte-erreichbaren Objekten zwischen p und q gibt.



- Bemerkung

- Dichte-Erreichbarkeit erzeugt einen Kernbereich und einen Rand

Algorithmus

- Aus der Definition ergibt sich unmittelbar ein Algorithmus zum Finden **des dichtebasierten Clusterings** einer Objektmenge O

```
clusterCount := 1;
for i from 1 to |O| do // Alle Punkte ansehen
  o := O.get(i);
  if o.clusterID = NULL and // Punkt in keinem Cluster
    kernobjekt(o) then // ... und im Kernbereich
    // eines (neuen) Clusters

    if expCl( o, O, clusterCount ) != 0 // Cluster berechnen
    then clusterCount++; // Nächster Cluster
```

Analyse

- Benötigt – oberflächlich gesehen – nur einen Lauf durch die Daten
- Aber: **ExpandiereCluster ist teuer**
 - Sucht wiederholt nach allen Punkten in der ε -Nachbarschaft von Punkten o
 - Ohne multidimensionalen Index
 - Alle paarweisen Distanzen vorberechnen (Distanzmatrix – teuer)
 - Bei Anfrage für o : Alle Objekte p verwerfen mit $p \notin N_\varepsilon(o)$
 - Benötigt $O(n^2)$ Zeit und Platz - schlecht
 - **Mit multidimensionalem Index**
 - Multidimensionaler Index muss Nachbarschaftsqueries unterstützen
 - Damit: $O(n^*)$ Aufwand für eine ε -Query)
- Gleiches Problem beim Test `kernobjekt()`

Inhalt dieser Vorlesung

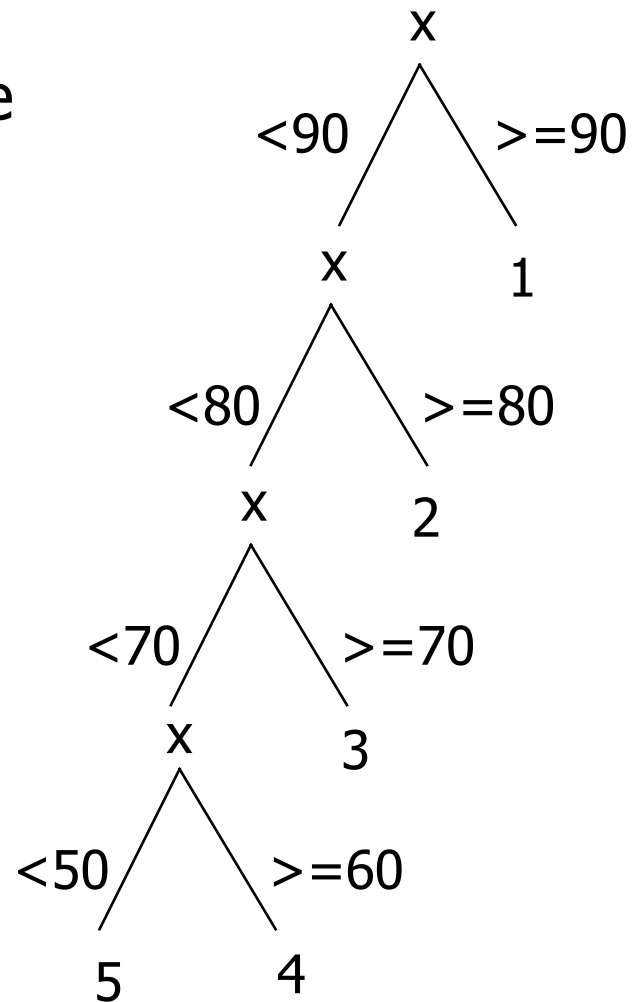
- Einführung
 - Problemstellung
 - Regression
 - Overfitting
- Naive-Bayes
- Entscheidungsbäume

Klassifikation

- Grundproblem
 - Gegeben eine Menge von Objekten mit Attributen (Features)
 - Gegeben eine Menge C von Klassen
 - Finde eine Klassifikationsfunktion $f:O \rightarrow C$, die **jedes Objekt aus O (s)einer Klasse c_i** zuordnet
- Wie soll das gehen?
 - Zusätzlich ist eine Trainingsmenge T mit $f':T \rightarrow C$ gegeben
 - „**Supervised learning**“
- Klassifikation und **Vorhersage**
 - Oftmals sind T Daten aus der Vergangenheit
 - Z.B. Kunden, deren Kredite geplatzt sind
 - O sind aktuelle Daten, deren Klassenzugehörigkeit man vorhersagen will
 - Z.B. Neue Kunden, die einen Kredit beantragen

Manchmal ganz einfach

- Aufgabe: Weise Studenten ihre Note (Klasse) aufgrund ihrer Punktzahl (Feature) zu
- Klassenzugehörigkeit ist auf dem Attribut definiert
 - Das ist idR leider nicht so

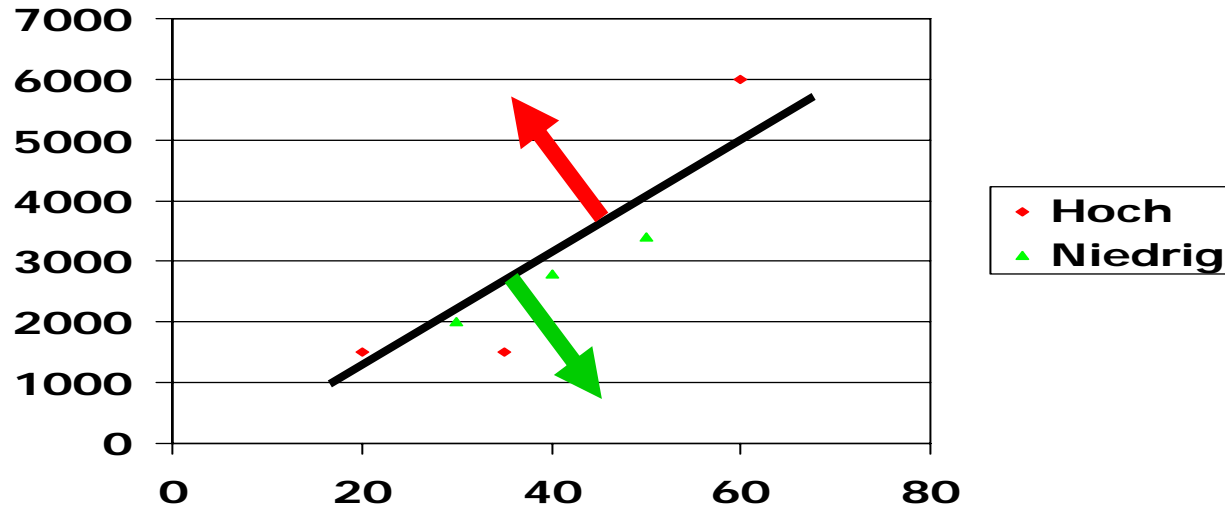


Etwas komplexer

| ID | Alter | Einkommen | Risiko |
|----|-------|-----------|---------|
| 1 | 20 | 1500 | Hoch |
| 2 | 30 | 2000 | Niedrig |
| 3 | 35 | 1500 | Hoch |
| 4 | 40 | 2800 | Niedrig |
| 5 | 50 | 3000 | Niedrig |
| 6 | 60 | 6000 | Hoch |

- Welches Risiko schätzen wir für eine Person von 45 Jahren mit 4000 Euro Einkommen?

Regression



- Punkte auftragen, Klasse durch Farbe gekennzeichnet
- **Regressionsgerade** berechnen (Fehlerminimierung)
 - Lineare Regression
- Zählen, auf welcher Seite welche Klasse in der Überzahl ist
- Als **Entscheidungsfunktion** benutzen
 - Linke-über der Gerade: Risikoklasse hoch (niedrig)
 - Rechts-unter der Gerade: Risikoklasse niedrig (hoch)

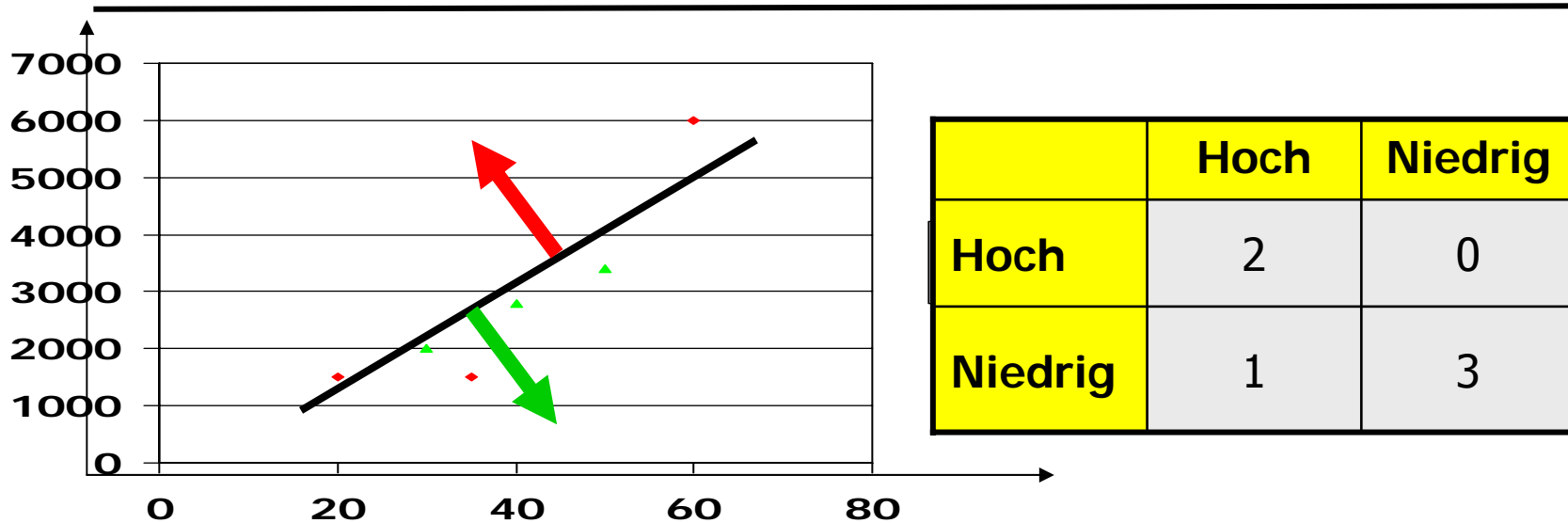
Bewertung einer Klassifikation

- Jeder (nicht-triviale) Klassifikator macht Fehler
- Binäres Entscheidungsproblem: „Kunde hat Risikoklasse hoch“

| | Real: Hoch | Real: Niedrig |
|------------------------|----------------|----------------|
| Klassifikator: Hoch | true-positive | false-positive |
| Klassifikator: Niedrig | false-negative | true-negative |

- **Precision** = $TP/(TP+FP)$
 - Wie viele der als hoch eingeschätzten Kunden haben tatsächliches hohes Risiko?
- **Recall** = $TP/(TP+FN)$
 - Wie viele der Kunden mit hohem Risiko haben wir als solche entdeckt?

Beispiel



- **Konfusionsmatrix**

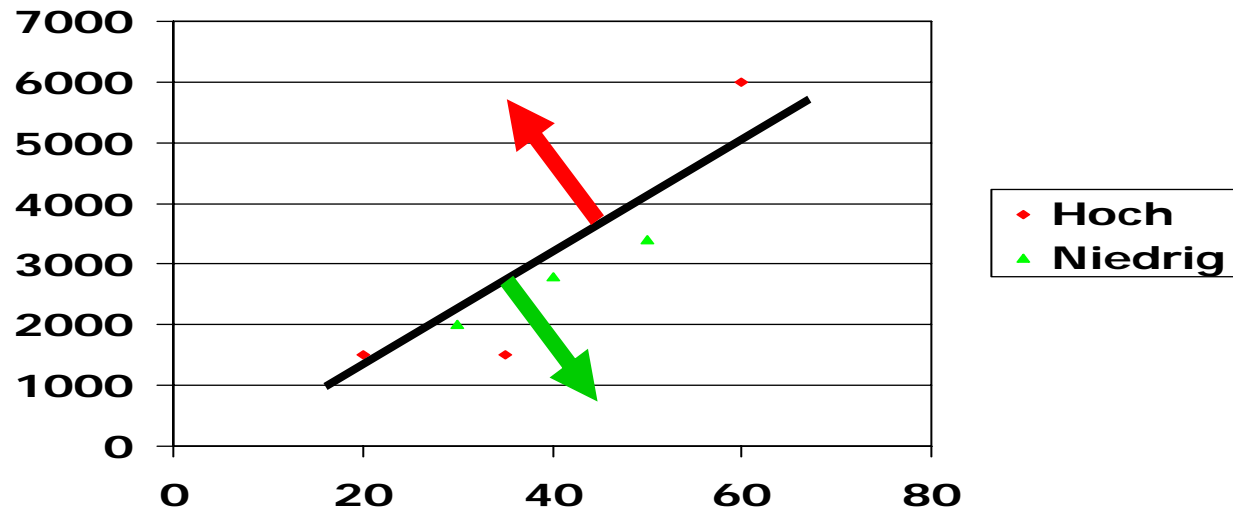
- Precision = $TP / (TP + FP) = 2/2$

 - Wenn wir hohes Risiko vorhersagen, haben wir immer Recht

- Recall = $TP / (TP + FN) = 2/3$

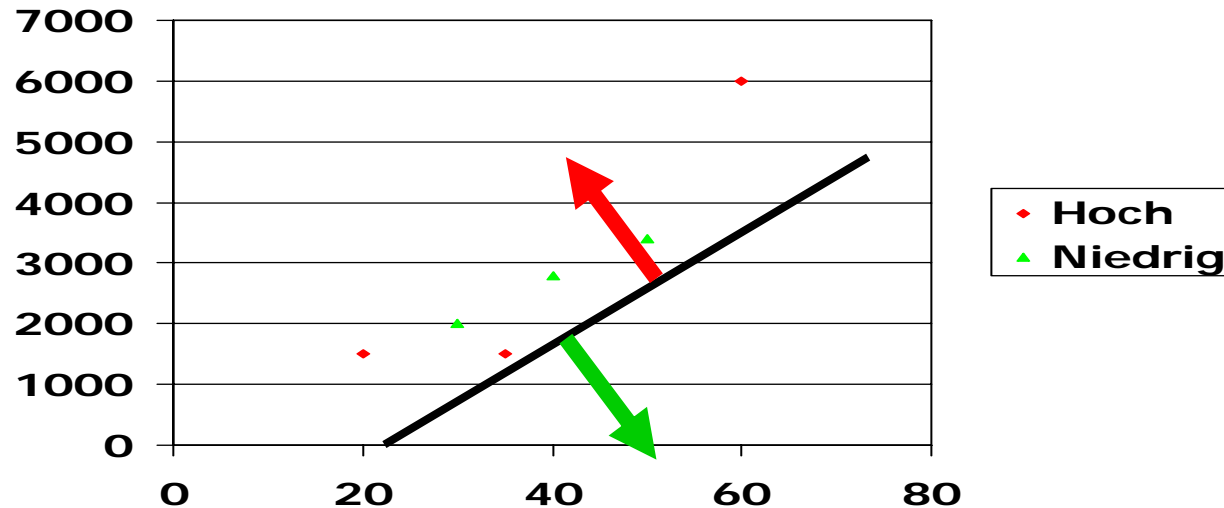
 - Aber wir verfehlen leider 33% aller hohen Risiken

Verbesserung



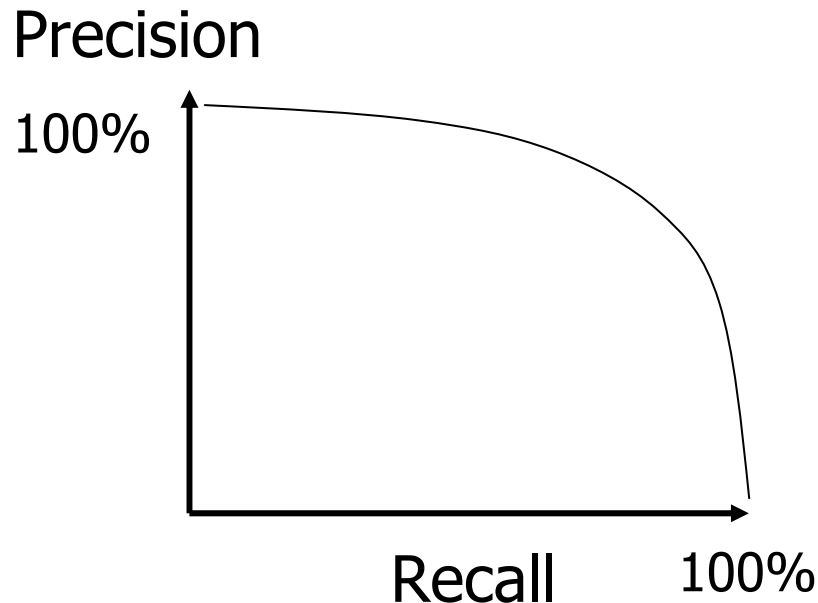
- Was können wir tun?

Verbesserung



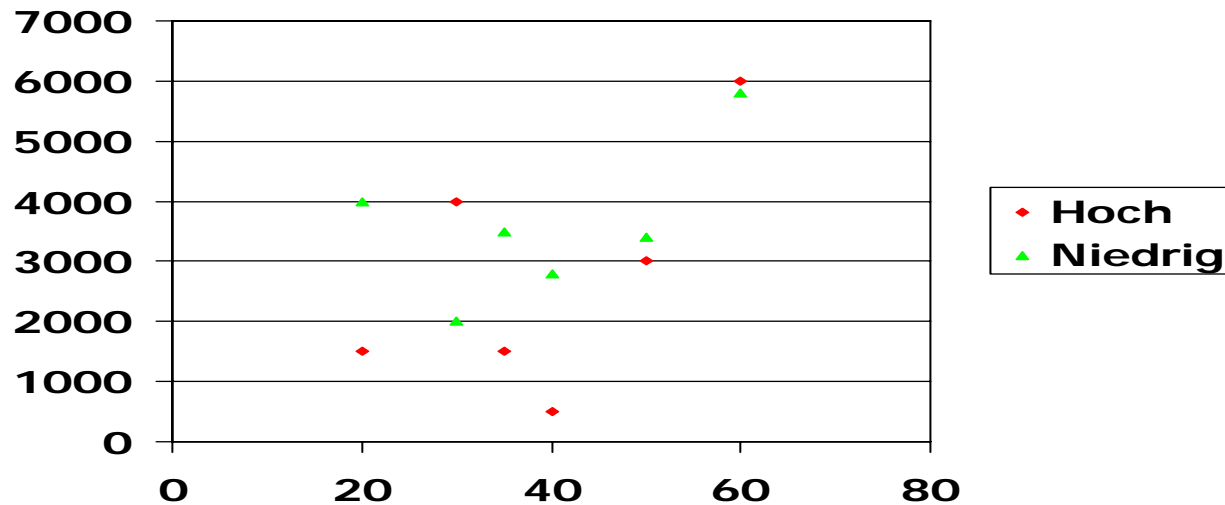
- Was können wir tun?
- Verschieben der Gerade
 - Nach unten: Erhöhter Recall, verringerte Precision
 - Nach oben: Verringerter Recall, erhöhte Precision

Area under the Curve, ROC



- Fast alle Klassifikationsalgorithmen haben einen **Parameter**, an dem man zwischen Precision und Recall verschieben kann
- Welcher Faktor wichtiger ist – anwendungsabhängig
 - Konservative Bank: Bloß keine Risiken, lieber mal einen Kredit ablehnen
 - Aggressive Bank: Lieber mal ein Risiko eingehen, als Kredite ablehnen

Regression – viele Probleme



- (Lineare) Regression ist eine sehr einfache Methode
- Funktioniert nur bei numerischen, vollständig geordneten Attributen
 - Aber auch mit vielen Dimensionen
- Nimmt immer einen funktionalen Zusammenhang an und versucht diesen zu finden
- Viele Erweiterungen für andere Funktionsklassen
 - Die auf zunehmend komplizierter zu lösende Gleichungssysteme führen

Kategorielle Attribute

| ID | Alter | Autotyp | Risiko |
|----|-------|---------|---------|
| 1 | 23 | Familie | hoch |
| 2 | 17 | Sport | hoch |
| 3 | 43 | Sport | hoch |
| 4 | 68 | Familie | niedrig |
| 5 | 32 | LKW | niedrig |

- **Klassenzugehörigkeit** hängt von zwei Attributen ab
- Nehmen wir an, die Klassen wurden von einem Sachbearbeiter zugewiesen. Was hatte der vielleicht im Kopf?
 - if Alter > 50 then Risikoklasse = Niedrig
 - elseif Alter < 32 then Risikoklasse = Hoch
 - elseif Autotyp = Sport then Risikoklasse = Hoch
 - else Risikoklasse = Niedrig

Entscheidungsregeln

| ID | Alter | Autotyp | Risiko |
|----|-------|---------|---------|
| 1 | 23 | Familie | hoch |
| 2 | 17 | Sport | hoch |
| 3 | 43 | Sport | hoch |
| 4 | 68 | Familie | niedrig |
| 5 | 32 | LKW | niedrig |

- Geht das mit **weniger Entscheidungen**?
 - if Alter>50 then Risikoklasse = Niedrig
 - elseif Autotyp=LKW then Risikoklasse = Niedrig
 - else Risikoklasse = Hoch

Noch mal

| ID | Alter | Autotyp | Risiko |
|----|-------|---------|---------|
| 1 | 23 | Familie | hoch |
| 2 | 17 | Sport | hoch |
| 3 | 43 | Sport | hoch |
| 4 | 68 | Familie | niedrig |
| 5 | 32 | LKW | niedrig |

- Warum nicht

- If Alter=17 and Autotype=Sport then Risikoklasse = hoch
- elseif Alter=23 and Autotype=Familie then Risikoklasse = hoch
- elseif Alter=32 and Autotype=LKW then Risikoklasse = hoch
- elseif Alter=43 and Autotype=Sport then Risikoklasse = niedrig
- else Risikoklasse = niedrig

Overfitting

- Klassifikation versucht immer, aus den beobachteten Daten **ein Modell abzuleiten**
 - Z.B. Regeln über Wertebereiche
- Man sieht aber immer nur einen **Ausschnitt** der Welt
 - Ausschnitt der Objekte (Trainingsdaten) und Ausschnitt aller möglichen Feature
 - Je mehr, desto besser
- **Overfitting**
 - Wenn man seine Regeln zu eng an den Ausschnitt anlehnt, ist man zu abhängig von der (eventuell schlechten) Wahl der Trainingsdaten
 - **Mangelnde Generalisierung** der Regeln
 - Funktioniert super auf den Trainingsdaten, aber schlecht auf realen Daten

Was tun gegen Overfitting?

- Wenn man auf den Trainingsdaten lernt und von den Testdaten die Klassen nicht kennt – wie kann man dann den Grad an Overfitting beurteilen?
- **Cross-Validierung** (Leave-one-out)
 - **Teilen der Trainingsdaten** in k zufällige Partitionen
 - k -maliges Lernen auf $k-1$ Partitionen und Anwendung des Modells auf der k 'ten Partition
 - Die Ergebnisse (Precision, Recall) sollten sehr nahe beieinander liegen
 - Und die Ergebnisse sollten besser sein, je größer k – größere Trainingsmenge
 - Man nimmt meist den Mittelwert als **erwartete Performance** des Klassifikators auf neuen Daten

Und noch mal

| ID | Alter | Autotyp | Risiko |
|----|-------|---------|---------|
| 1 | 23 | Familie | hoch |
| 2 | 17 | Sport | hoch |
| 3 | 43 | Sport | hoch |
| 4 | 68 | Familie | niedrig |
| 5 | 32 | LKW | niedrig |

- Warum nicht
 - If ID=1 then Risikoklasse = hoch
elseif ID=2 then Risikoklasse = hoch
elseif ID=3 then Risikoklasse = hoch
elseif ID=4 then Risikoklasse = niedrig
else Risikoklasse = niedrig
- ID hat nichts mit dem Risiko zu tun
- Das wissen wir – der Computer weiß es nicht
- **Feature Auswahl** ist essentiell
 - ID wäre so oder so ein schlechtes Feature – Warum?

Inhalt dieser Vorlesung

- Einführung
- Naive-Bayes Klassifikator
- Entscheidungsbäume

Bayes'sche Klassifikation

- Wahrscheinlichkeitsbasiertes, **einfaches Verfahren**
 - Das meistens erstaunlich gut funktioniert
- Gegeben
 - Klassenmenge C , Trainingsmenge O
 - Menge F von Features: **binäre Eigenschaften** aller Objekte $o \in O$
 - Z.B. Attribut $A[o]=X$
- Wir suchen für ein Objekt o **die Wahrscheinlichkeiten** $p(c_i|o)$ der Zugehörigkeit zu jeder Klasse $c_i \in C$
- Klassifikationsergebnis ist die Klasse, die die **höchste Wahrscheinlichkeit** hat
- Objekte werden durch Feature $F=\{f_1, \dots, f_n\}$ beschrieben

$$p(c | o) = p(c | f_1[o], \dots, f_n[o]) = p(c | f_1, \dots, f_n)$$

Wahrscheinlichkeiten

- Gesucht: $p(c|o)$
- Wir haben
 - O mit binären Features f_1, \dots, f_n
 - Die **A-Priori Wahrscheinlichkeit** $p(f)$ jedes Features f
 - Bei vielen Objekten ist f wahr?
 - Die **A-Priori Wahrscheinlichkeiten** $p(c)$ aller Klassen $c \in C$
 - Wie viele Objekte aus O gehören zu jeder Klasse?
 - Die **bedingten Wahrscheinlichkeiten** $p(f|c)$, dass Feature f bei Klasse c wahr ist
 - Wie oft ist f bei allen Objekten der Klasse c wahr?
- Wir formulieren um und wenden Bayes' Theorem an

$$p(c | f_1, \dots, f_n) = \frac{p(f_1, \dots, f_n | c) * p(c)}{p(f_1, \dots, f_n)} \approx p(f_1, \dots, f_n | c) * p(c)$$

Naive Bayes

- Wir haben also $p(c | o) \approx p(f_1, \dots, f_n | c) * p(c)$
- Den ersten Term kann man für **hochdimensionale Featureräume** nicht vernünftig berechnen
 - Es gibt zu viele Kombinationen (2^n) von Features
 - Achtung: Ein Attribut mit 10 Werten ergibt potentiell 10 Feature
 - Jede einzelne Kombination ist in jedem realen Trainingsset sehr selten
- „Naive“ Annahme: **Statistische Unabhängigkeit** aller Feature
 - Denn dann gilt $p(f_1, \dots, f_n | c) = p(f_1 | c) * \dots * p(f_n | c)$
- Und damit

$$p(c | o) \approx p(c) * \prod_{i=1}^n p(f_i | c)$$

Beispiel

- Wir raten binäre Features

- Jung (Alter<25)?
- Sportwagen?

- Damit

- $P(c=\text{hoch})=3/5$
- $p(f_1=\text{jung})=2/5$
- $P(f_2=\text{sportwagen})=2/5$
- $P(\text{jung}|\text{hoch})=2/3$, $p(\text{jung}|\text{niedrig})=0$
- $P(\text{sportwagen}|\text{hoch})=2/3$, $p(\text{sportwagen}|\text{niedrig})=0$
- **Smoothing**: $p(\text{jung}|\text{niedrig})= p(\text{sportwagen}|\text{niedrig})=0,01$

- **Neuer Kunde**, 24, LKW

- $p(\text{hoch}|\text{jung}, \neg\text{sportwagen})=p(\text{hoch})*p(\text{jung}|\text{hoch})*(1-p(\text{sportwagen}|\text{hoch}))=3/5*2/3*1/3 = 6/45$
- $p(\text{niedrig}|\text{jung}, \neg\text{sportwagen})=2/5*0.01*0.01$

- **Neuer Kunde**, 35, Familienwagen

- $p(\text{hoch}|\text{alt}, \neg\text{sportwagen})=3/5*1/3*1/3 = 3/45$
- $p(\text{niedrig}|\text{alt}, \neg\text{sportwagen})=2/5*0.99*0.99$

| ID | Alter | Autotyp | Risiko |
|----|-------|---------|---------|
| 1 | 23 | Familie | hoch |
| 2 | 17 | Sport | hoch |
| 3 | 43 | Sport | hoch |
| 4 | 68 | Familie | niedrig |
| 5 | 32 | LKW | niedrig |

SQL

- Wir brauchen $p(f_i|c_j)$, $p(c_j)$, $p(f_i)$
- Wie kriegen wir die **effizient mit SQL?**
 - Schema: `obj(oid, fid, fvalue)`, `class(oid, cid)`

- $p(c_j)$

```
SELECT  cid, count(*)
FROM    class
GROUP BY cid;
```

- $p(f_i)$

```
SELECT  fid, count(*)
FROM    object
WHERE   fvalue = ?
GROUP BY fid;
```

- $p(f_i|c_j)$

```
SELECT  cid, fid, count(*)
FROM    object o, class c
WHERE   o.oid=c.oid AND
        fvalue = ?
GROUP BY cid, fid;
```

Fazit

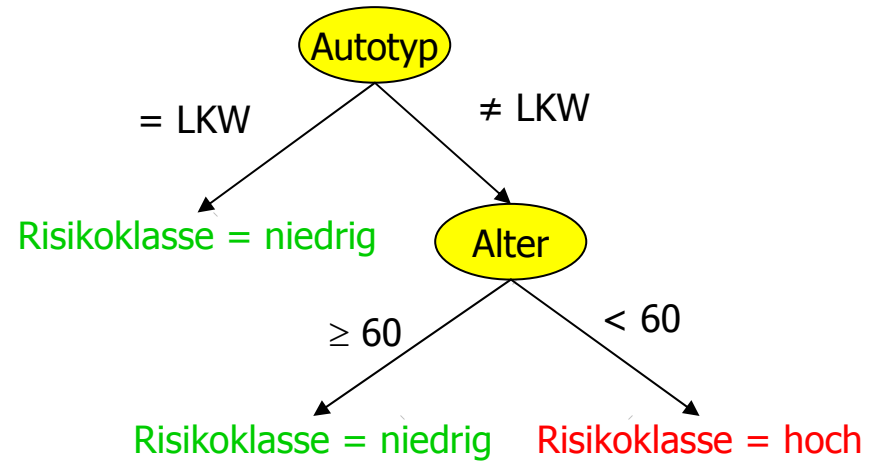
- Einfaches Verfahren
- Parameter können **schnell berechnet** werden
- Neue Trainingsdaten können **inkrementell** hinzugefügt werden
 - Z.B. Verwendung inkrementell aktualisierter materialisierter Sichten
- Platzbedarf vernachlässigbar
 - Hängt nicht von der Anzahl Objekte, sondern nur von der **Anzahl der Features und Klassen** ab
- Meist recht gute und robuste Ergebnisse
- Nachteil
 - Geringe Erklärungskraft
 - **Unabhängigkeitsannahme** stimmt meistens nicht
 - Abhilfe: Bayes'sche Netzwerke
 - Schwierige Featurewahl
 - Was tun bei **nahezu gleich-wahrscheinlichen Klassen**?

Inhalt dieser Vorlesung

- Einführung
- Naive-Bayes Klassifikator
- **Entscheidungsbäume**
 - Grundprinzip
 - ID3 und CART
 - Skalierbarkeit: SPRINT
 - Tree Pruning

Entscheidungsbäume

| ID | Alter | Autotyp | Risiko |
|----|-------|---------|---------|
| 1 | 23 | Familie | hoch |
| 2 | 17 | Sport | hoch |
| 3 | 43 | Sport | hoch |
| 4 | 68 | Familie | niedrig |
| 5 | 32 | LKW | niedrig |



- Definition

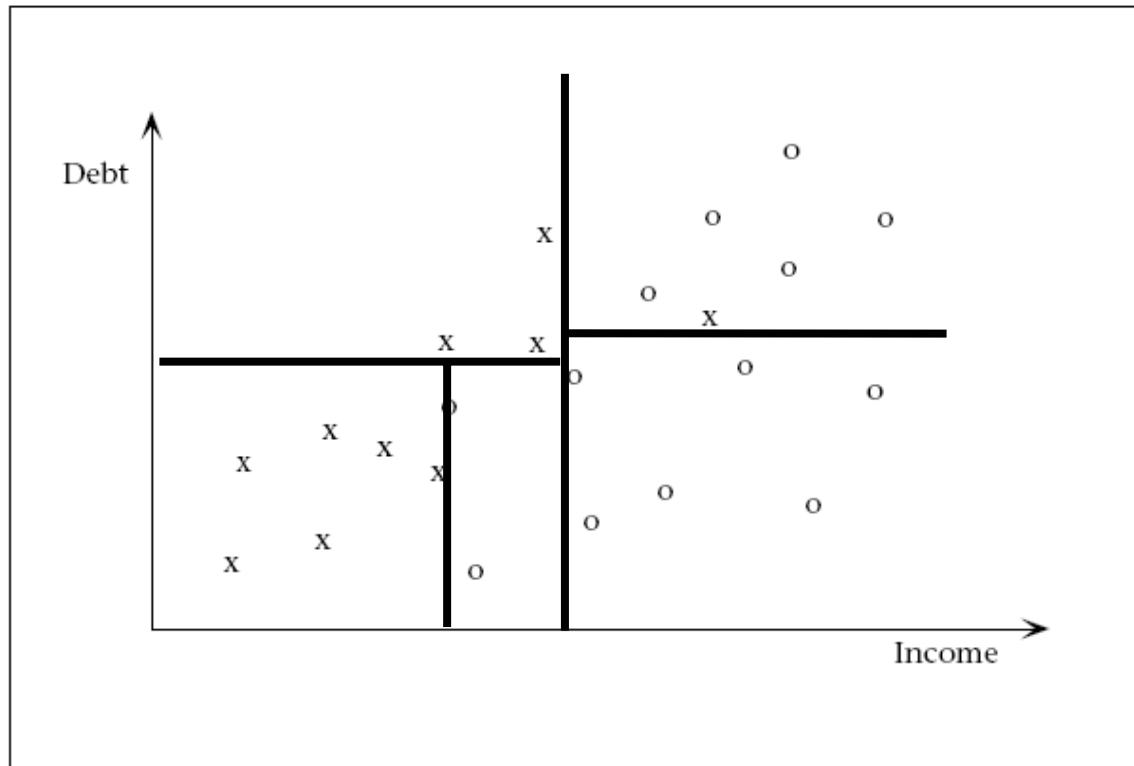
Ein *Entscheidungsbaum* ist ein Baum mit

- Jeder innere Knoten ist mit einem Attribut beschriftet
- Die ausgehenden Kanten eines inneren Knoten sind mit *Bedingungen an die Werte des Vaterknotens* beschriftet, deren Wertebereiche sich nicht überschneiden
- Jedes Blatt ist mit einer Klasse beschriftet
- Die Bedingungen einer Kante e widersprechen nie der Konjunktion der Bedingungen des Pfades von Root zum Vater von e

Anwendung und Berechnung

- Anwendung zur **Klassifikation** ist offensichtlich
 - Gegeben neues Objekt o
 - Laufe den Baum ab und beschreibe an jedem Knoten der Kante, deren Bedingung in o zutrifft
 - Wird ein Blatt mit Label c erreicht: Ordne o der Klasse c zu
 - Vorsicht: Ein klassen-eindeutiger DT ist i.A. nicht möglich
 - Also: Entscheidungsbäume dürfen **Fehler machen**
- Das interessantere Problem
 - Ableitung eines „guten“ Entscheidungsbaums aus den Daten
 - **Decision Tree Induction (DTI)**

Partitionierung



- Entscheidungsbäume **partitionieren den Raum** rekursiv

DT: Pro und Contra

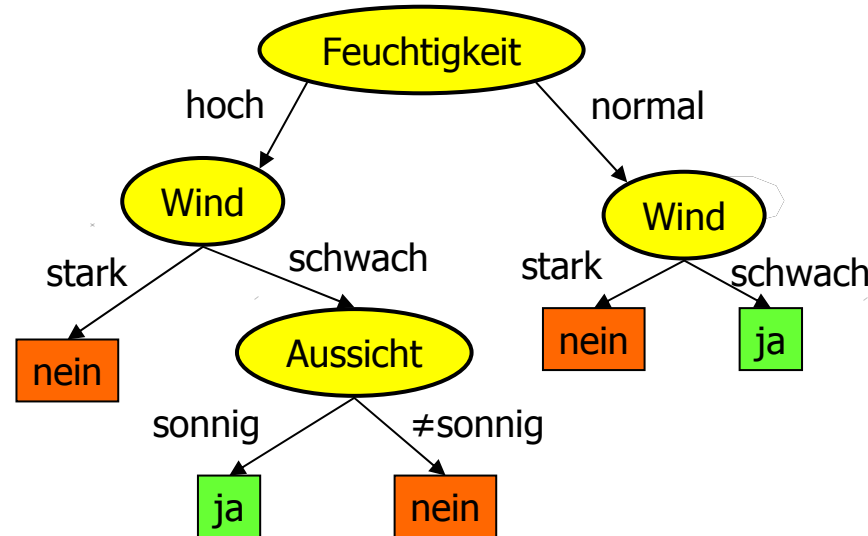
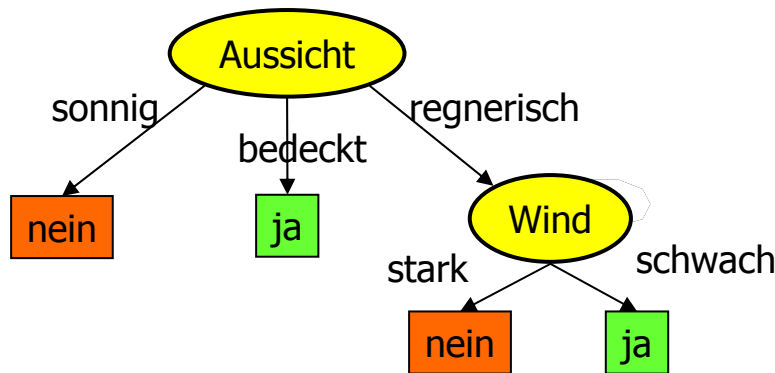
- Vorteile
 - **Verständliche Darstellung** – Menschen lieben DT
 - Leichte Ableitung von Regelmengen möglich
 - Schnelle Klassifikation (Baumhöhe ist $\sim O(\log(n))$)
 - Effiziente Verfahren für DTI bekannt
- Nachteile
 - Schnelle DTI-Algorithmen berechnen nur **rechwinklige Partitionierung** des Raums
 - Gefahr des **Overfitting** (siehe Tree Pruning)

DTI: Grundverfahren

- Gegeben Trainingsmenge T
 - Wähle ein Attribut A und Bedingungen (Splits) B_1, \dots, B_m
 - Für binäre Entscheidungsbäume: $m=2$
 - Partitioniere T gemäß Bedingungen in m Partitionen
 - Das erzeugt einen inneren Knoten im Baum
 - Wende Verfahren **rekursiv für jede der Partitionen** an
 - Solange nicht alle Objekte einer Partition der gleichen Klasse angehören oder ein anderes Stopp-Kriterium erfüllt ist
- Zentrale Frage: **Welches Attribut? Welche Splits?**

Auswirkungen

| Tag | Aussicht | Temperatur | Feuchtigkeit | Wind | Tennispielen |
|-----|------------|------------|--------------|---------|--------------|
| 1 | sonnig | heiß | hoch | schwach | nein |
| 2 | sonnig | heiß | hoch | stark | nein |
| 3 | bedeckt | heiß | hoch | schwach | ja |
| 4 | regnerisch | mild | hoch | schwach | ja |
| 5 | regnerisch | kühl | normal | schwach | ja |
| 6 | regnerisch | kühl | normal | stark | nein |
| 7 | ... | ... | ... | ... | ... |



Wahl der Splits

- Arten von Splits
 - Numerische Attribute: Wertebereiche
 - Kategoriale Attribute: Gleichheit, Mengenzugehörigkeit
- Zwei Optimierungsstrategien
 - Global: Splitte so, dass der DT insgesamt ...
 - die minimale Menge innerer Knoten hat
 - die minimale Baumtiefe hat
 - die minimale Menge von Fehlern erzeugt
 - ...
 - Alle zu teuer
 - Lokal: Splitte Partition so, dass ein **lokales Kriterium für die aktuelle (weiter zu splittende) Partition** optimiert wird
 - Ziel: Erhöhung der „Reinheit“ der Folgepartitionen

ID3 etc.: Entropy [Qui86, Qui93]

- Klassische DTI Algorithmen: ID3 / CART / C4.5 / C5.0
- Ziel: Anzahl der **erwarteten Vergleiche** minimieren
 - Also Baum balanciert und niedrig halten
 - Heuristik dazu: Wähle Split so, dass die **Entropie der neuen Partitionen maximal kleiner** wird

- Definition

Die **Entropie** einer Trainingsmenge T bzgl. Klassen $C=\{c_1, \dots, c_n\}$ ist definiert als

$$\text{entropy}(T) = -\sum_{i=1}^n p_i * \log(p_i)$$

mit $p_i =$ Häufigkeit der Klasse c_i in T

- Bemerkung

- Entropie misst die **Reinheit** von T bzgl. C
 - $\text{entropy}(T) = 1$: maximale Verwirrung
 - $\text{entropy}(T) = 0$: homogene Menge (ein $p_i=1$, alle anderen $=0$)

Information Gain

- Die Entropie soll durch die Partitionierung maximal verringert werden

- Andersherum: Der Information Gain soll maximal sein

- Definition

Der *Information Gain* einer Partitionierung P (also eines Splits) einer Grundmenge T mit $|P|=m$ ist definiert als

$$\text{gain}(T, P) = \text{entropy}(T) - \sum_{i=1}^m \text{entropy}(P_i)$$

Der *gewichtete Information Gain* ist definiert als

$$\text{wgain}(T, P) = \text{entropy}(T) - \sum_{i=1}^m \frac{|P_i|}{|T|} \text{entropy}(P_i)$$

Beispiel

| age | income | student | credit_rating | buys_computer |
|---------|--------|---------|---------------|---------------|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| 31...40 | high | no | fair | yes |
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| 31...40 | low | yes | excellent | yes |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |
| >40 | medium | yes | fair | yes |
| <=30 | medium | yes | excellent | yes |
| 31...40 | medium | no | excellent | yes |
| 31...40 | high | yes | fair | yes |
| >40 | medium | no | excellent | no |

Quelle:
[HK06]

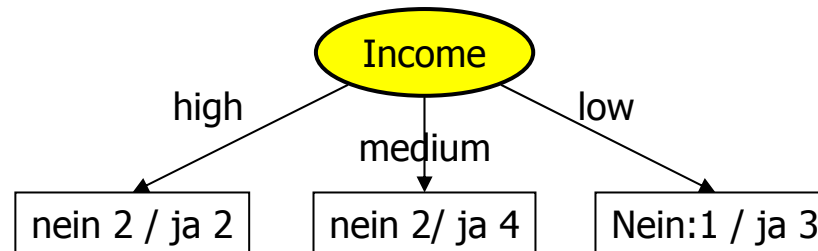
$$\text{entropy}(T) = -\frac{9}{14} \log_2\left(\frac{9}{14}\right) - \frac{5}{14} \log_2\left(\frac{5}{14}\right) = 0.940$$

Splitpunkte

| age | income | student | credit_rating | buys_computer |
|---------|--------|---------|---------------|---------------|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| 31...40 | high | no | fair | yes |
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| 31...40 | low | yes | excellent | yes |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |
| >40 | medium | yes | fair | yes |
| <=30 | medium | yes | excellent | yes |
| 31...40 | medium | no | excellent | yes |
| 31...40 | high | yes | fair | yes |
| >40 | medium | no | excellent | no |

- Split bei **income**

- Drei Partitionen „high“, „medium“, „low“
- $\text{entropy}(\text{high}) = -2/4 \cdot \log(2/4) - 2/4 \cdot \log(2/4) \sim 1$
- $\text{entropy}(\text{medium}) = -2/6 \cdot \log(2/6) - 4/6 \cdot \log(4/6) \sim 0.91$
- $\text{entropy}(\text{low}) = -3/4 \cdot \log(3/4) - 1/4 \cdot \log(1/4) \sim 0.81$
- $\text{wgain}(\text{income}) = 0.94 - (4/14 \cdot 1 + 6/14 \cdot 0.91 + 4/14 \cdot 0.81) \sim 0.3$

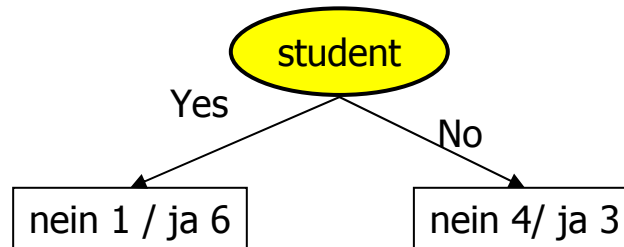


Splitpunkte

| age | income | student | credit_rating | buys_computer |
|---------|--------|---------|---------------|---------------|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| 31...40 | high | no | fair | yes |
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| 31...40 | low | yes | excellent | yes |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |
| >40 | medium | yes | fair | yes |
| <=30 | medium | yes | excellent | yes |
| 31...40 | medium | no | excellent | yes |
| 31...40 | high | yes | fair | yes |
| >40 | medium | no | excellent | no |

- Split bei **student**

- Zwei Partitionen
- $\text{entropy}(\text{yes}) = -6/7 \cdot \log(6/7) - 1/7 \cdot \log(1/7) \sim 0.59$
- $\text{entropy}(\text{no}) = -3/7 \cdot \log(3/7) - 4/7 \cdot \log(4/7) \sim 0.98$
- $\text{wgain}(\text{student}) = 0.94 - (7/14 \cdot 0.59 + 7/14 \cdot 0.98) \sim 0.15$



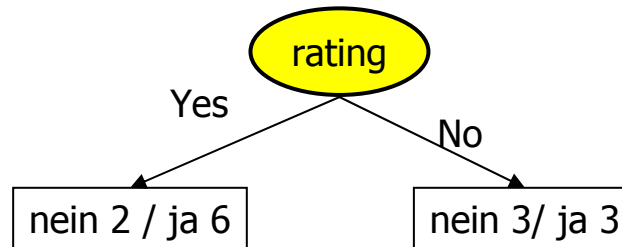
- Also: **Lieber bei income als bei student splitten**

Splitpunkte

| age | income | student | credit_rating | buys_computer |
|---------|--------|---------|---------------|---------------|
| <=30 | high | no | fair | no |
| <=30 | high | no | excellent | no |
| 31...40 | high | no | fair | yes |
| >40 | medium | no | fair | yes |
| >40 | low | yes | fair | yes |
| >40 | low | yes | excellent | no |
| 31...40 | low | yes | excellent | yes |
| <=30 | medium | no | fair | no |
| <=30 | low | yes | fair | yes |
| >40 | medium | yes | fair | yes |
| <=30 | medium | yes | excellent | yes |
| 31...40 | medium | no | excellent | yes |
| 31...40 | high | yes | fair | yes |
| >40 | medium | no | excellent | no |

- Split bei **credit_rating**

- Zwei Partitionen
- $\text{entropy}(\text{fair}) = -2/8 \cdot \log(2/8) - 6/8 \cdot \log(6/8) \sim 0.81$
- $\text{entropy}(\text{excellent}) = -3/6 \cdot \log(3/6) - 3/6 \cdot \log(3/6) \sim 1$
- $\text{wgain}(\text{rating}) = 0.94 - (8/14 \cdot 0.81 + 6/14 \cdot 1) \sim 0.04$



- Also: **Lieber bei income als bei student als bei rating splitten**

Berechnung des besten Split

- Gegeben Trainingsmenge T
 - Für alle Attribute A_i
 - Für alle Partitionierungen P_{ij}
 - Berechne $wgain(A_i, P_{ij})$
 - Wähle den Split (A_i, P_{ij}) , für den $wgain()$ am größten ist
 - Partitioniere T gemäß entlang A_i in Partitionen P_{ij}
 - Wende Verfahren **rekursiv für jede der Partitionen** an
- Probleme
 - Die **Doppelschleife ist sehr teuer** für Attribute mit vielen Werten / numerische Attribute
 - Man muss theoretisch jede mögliche Aufteilung der Werte in beliebige viele Partitionen betrachten

Typischer Trick: Nur binäre Splits

- Sei $j = \max(|A_i|)$, n Attribute, m Tupel
- Betrachte nur **binäre Splits der sortierten Werte** als mögliche Splits
 - Für total geordnete Wertebereiche; sonst 2^j mögliche Aufteilungen
- Immer noch teuer: Man muss **für jeden inneren Knoten**
 - $O(n*j*\log(j))$ Vergleiche zur Sortierung
 - Pro Attribut $O(j)$ mögliche Splits
 - Pro Split alle Tupel (der Partition) scannen, um $w_{\text{gain}}()$ auszurechnen
 - Kann man inkrementell implementieren (wechselnde Werte)
 - Zusammen: $O(n*j*\log(j) + n*j*m)$
 - Dabei ändert sich
 - „ n “ wird pro Split um eins kleiner
 - Wenn wir pro Attribut nur einmal splitten wollen – das ist nicht klar
 - „ j “ bleibt gleich
 - m wird pro Split wesentlich kleiner
 - Jeder Split partitioniert m
 - Die Tupel bleiben aber in einer Tabelle (Zugriff über Index?)

Inhalt dieser Vorlesung

- Einführung
- Naive-Bayes Klassifikator
- Entscheidungsbäume
 - Grundprinzip
 - ID3 und CART
 - Skalierbarkeit: **SPRINT**
 - Tree Pruning

Skalierbarkeit: SPRINT [SAM96]

- Bisherige Verfahren haben nur dann akzeptable Geschwindigkeit, wenn **alle Daten im Hauptspeicher** sind
 - Vor allem das dauernde Sortieren ist teuer
- Idee von SPRINT: Nur einmal sortieren
- Idee
 - Sortiere alle Werte v pro Attribut a und speichere diese Listen (I)
 - Pro Attribut: Alle Tripel $(v, TID, \text{class}(TID))$ mit $a[TID]=v$ in eine Liste I_a
 - Bei Split in a : Teile I_a am Splitpunkt in zwei Listen
 - Aufteilung trivial für a
 - Andere Attributlisten linear durchlaufen und gemäß Zuordnung der Tupel zu Partitionen teilen
- Vorteil
 - Keine neuen Sortierungen
 - Häufigkeiten der Klassen können pro Liste vorgehalten werden
 - Schnellere Berechnung des Gains
 - Gute **Parallelisierbarkeit**: Alg. arbeitet nur auf den partitionierten Listen
 - Kaum Hauptspeicheranforderungen

Beispiel

| ID | Alter | Autotyp | Risiko |
|----|-------|---------|---------|
| 0 | 23 | Familie | hoch |
| 1 | 17 | Sport | hoch |
| 2 | 43 | Sport | hoch |
| 3 | 68 | Familie | niedrig |
| 4 | 32 | LKW | niedrig |
| 5 | 20 | Familie | hoch |

| Alter | Klasse | Id |
|-------|---------|----|
| 17 | Hoch | 1 |
| 20 | Hoch | 5 |
| 23 | Hoch | 0 |
| 32 | Niedrig | 4 |
| 43 | Hoch | 2 |
| 68 | Niedrig | 3 |

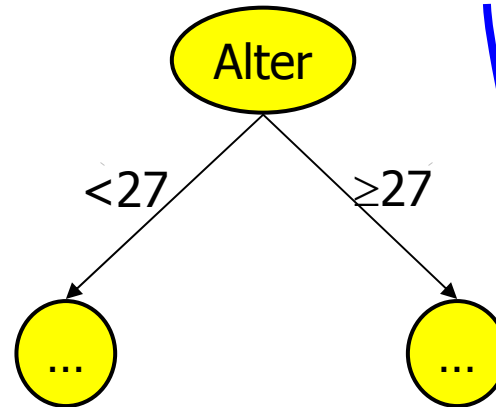
| Autotyp | Klasse | Id |
|---------|---------|----|
| Familie | Hoch | 0 |
| Familie | Niedrig | 3 |
| Familie | Hoch | 5 |
| LKW | Niedrig | 4 |
| Sport | Hoch | 1 |
| Sport | Hoch | 2 |

Quelle:
[ES00]

Einfach teilen

| Alter | Klasse | Id |
|-------|--------|----|
| 17 | Hoch | 1 |
| 20 | Hoch | 5 |
| 23 | Hoch | 0 |

| Autotyp | Klasse | Id |
|---------|--------|----|
| Familie | Hoch | 0 |
| Familie | Hoch | 5 |
| Sport | Hoch | 1 |



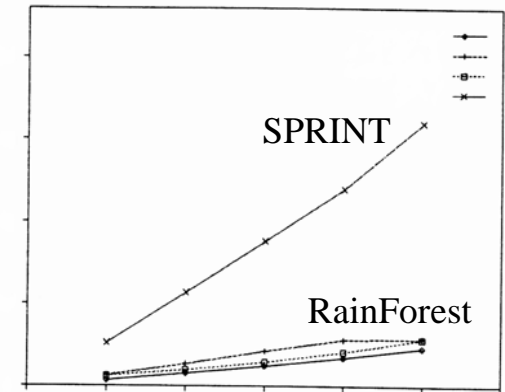
| Alter | Klasse | Id |
|-------|---------|----|
| 32 | Niedrig | 4 |
| 43 | Hoch | 2 |
| 68 | Niedrig | 3 |

| Autotyp | Klasse | Id |
|---------|---------|----|
| Familie | Niedrig | 3 |
| LKW | Niedrig | 4 |
| Sport | Hoch | 2 |

Einmal scannen und teilen

Weitere Algorithmen

- RainForest (VLDB'98)
 - Speicherung aggregierter Counts in den Attributlisten
 - Die können sehr effizient aus- und weitergerechnet werden
- BOAT (PODS'99)
 - "Bootstrapping"
 - Erzeugung von k (kleinen) Samples aus den Daten
 - Pro Sample wird im Hauptspeicher ein DT gebaut
 - Methode ist relativ egal
 - Aus den k Bäumen wird ein Konsensusbaum berechnet
 - Ergebnis: 2-3 mal schneller als Rainforest, sehr ähnliches Ergebnis



Inhalt dieser Vorlesung

- Einführung
- Naive-Bayes Klassifikator
- Entscheidungsbäume
 - Grundprinzip
 - ID3 und CART
 - Skalierbarkeit: SPRINT
 - Tree Pruning

Overfitting

- Recall: DTI passiert auf einer Trainingsmenge T , Anwendung erfolgt auf der Objektmenge O
- Ein DT T_1 **overfittet**, wenn es einen Baum T_2 gibt mit
 - T_1 ist auf T besser als T_2
 - T_2 ist auf O besser als T_1
 - Dann ist T_1 **zu spezifisch für T** und nicht mehr geeignet für O
- Overfitting passiert typischerweise in den **unteren Knoten** eines DT
 - Betrachtung immer kleinerer, speziellerer Tupelmengen
 - Entscheidungen haben bzgl. O keine statistische Signifikanz mehr
- Abhilfe: **Tree Pruning**

Tree Pruning

- Idee: Entferne (untere) Teilbäume, die
 - Nur noch wenig Tupel klassifizieren
 - Deren Abschneiden die **Gesamtfehlerrate auf T** nur wenig verschlechtern
 - Und auf O hoffentlich gar nicht
- „Kostenreduktionspruning“
 - Teile T in Trainingsmenge T1 und Testmenge T2
 - Lerne Baum auf T1
 - Entferne so lange Blattknoten von T1, wie der Fehler des Baums auf T2 kleiner wird
 - Problem: Benötigt viele Daten, Wahl von T1 / T2?
- Andere Möglichkeiten
 - Beobachtung: **Je kleiner der Baum, desto kleiner das Overfitting**
 - Also: Nimm die Baumgröße und den aktuellen Fehler in Entscheidung für Splits auf
 - Für zu anderen Splitheuristiken

Literatur

- Quinlan, J. (1986). "Induction of decision trees." *Machine Learning* **11**(1): 81-106.
- Quinlan, J. (1993). "C4. 5: Programs for Machine Learning". San Francisco, Morgan Kaufmann.
- Shafer, J. C., Agrawal, R. and Mehta, M. (1996). "SPRINT: A Scalable Parallel Classifier for Data Mining". 22th Conference on Very Large Databases, Mumbai, India.
- Ester, M. and Sander, J. (2000). "Knowledge Discovery in Databases". Berlin, Springer.
- Han, J. and Kamber, M. (2006). "Data Mining. Concepts and Techniques", Morgan Kaufmann.