Data Warehousing und Data Mining

Klassifikation

Ulf Leser

Wissensmanagement in der Bioinformatik

Inhalt dieser Vorlesung

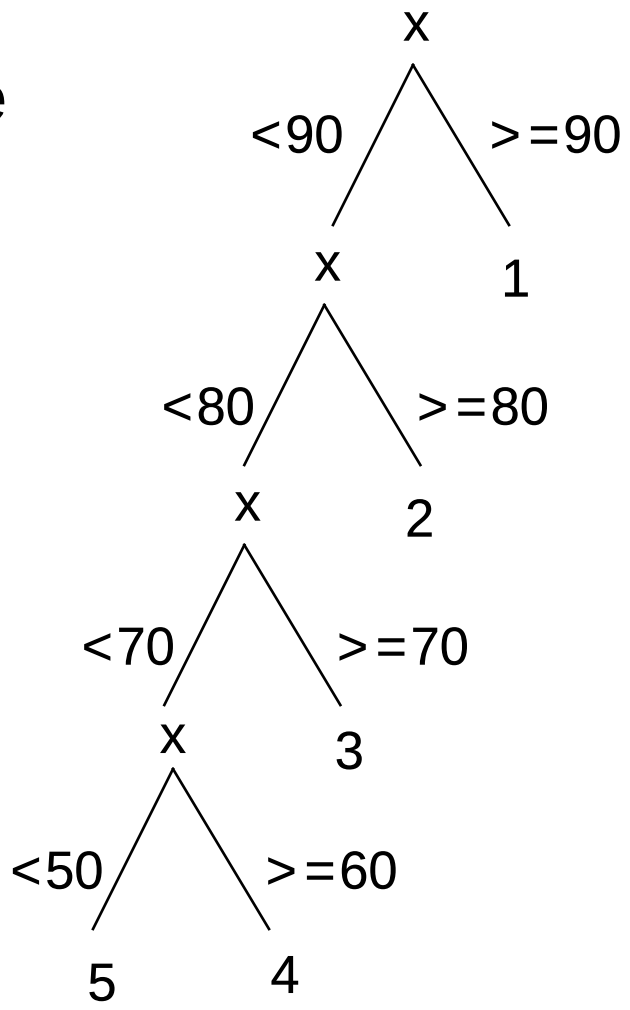
- Einführung
  - Problemstellung
  - Evaluation
  - Overfitting
- kNN Klassifikator
- Naive-Bayes
- Entscheidungsbäume

Klassifikation

- Grundproblem
  - Gegeben eine Menge von Objekten mit Attributen (Features)
  - Gegeben eine Menge C von Klassen
  - Finde eine Klassifikationsfunktion f:O→C, die jedes Objekt aus O einer Klasse c_i zuordnet
- Wie soll das gehen?
  - Lerne f aus einer Trainingsmenge T, für die f‘:T→C gegeben
  - Supervised learning
- Klassifikation und Vorhersage
  - T enthält Daten aus der Vergangenheit, deren Klasse man kennt
    - Z.B. Kunden, deren Kredite geplatzt sind / bedient wurden
  - O sind neue Daten, deren Klasse 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

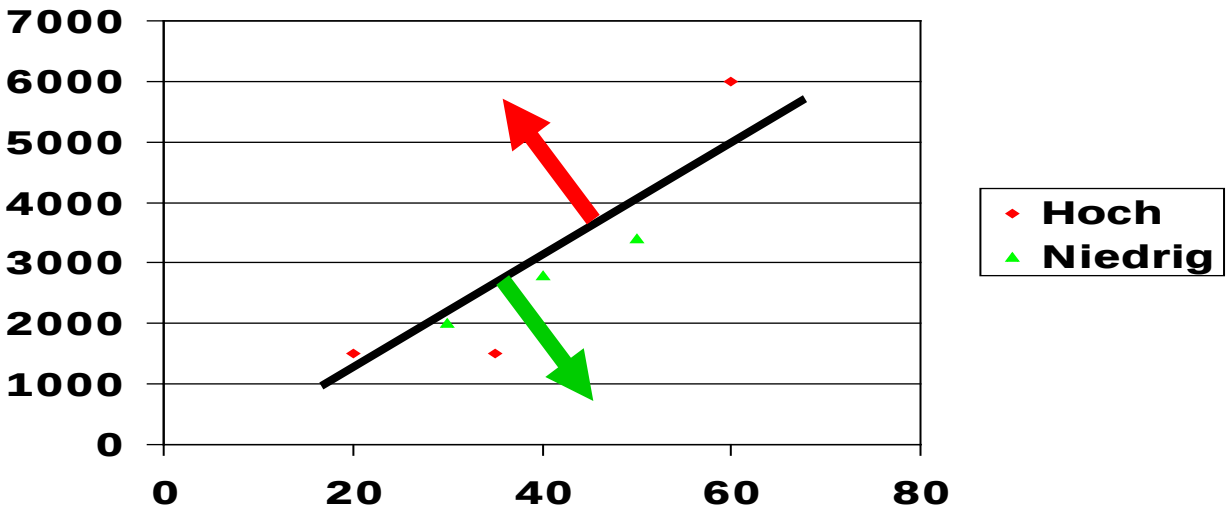


Etwas komplexer

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?

Beispiel: Lineare Trennung



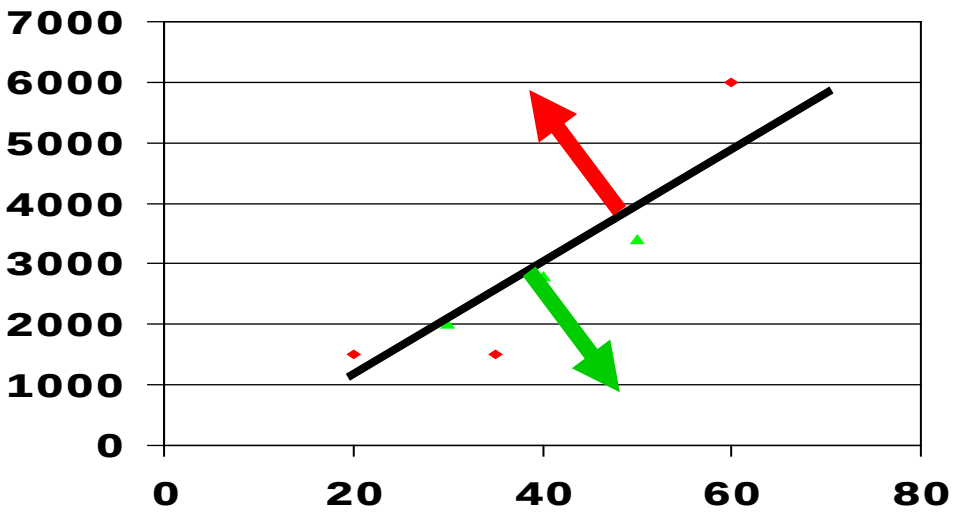
- (Lineare) Funktionen f auf den Attributwerten
- Klasse ergibt sich aus Lage zur Trennfunktion
  - Links-über der Gerade: Risikoklasse hoch (niedrig)
  - Rechts-unter der Gerade: Risikoklasse niedrig (hoch)
- Finde f, die den Fehler auf Trainingsdaten minimiert

Bewertung einer Klassifikation

- Precision = TP/(TP+FP)
  - Wie viele der als riskant eingeschätzten Kunden sind riskant?
- Recall = TP/(TP+FN)
  - Wie viele der Kunden mit hohem Risiko wurden entdeckt?
- Accuracy = (TP+TN) / (TP+FP+FN+TN)
  - Wie viele der Einschätzungen sind korrekt?
  - Wenig aussagekräftig bei großer Class Imbalance

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

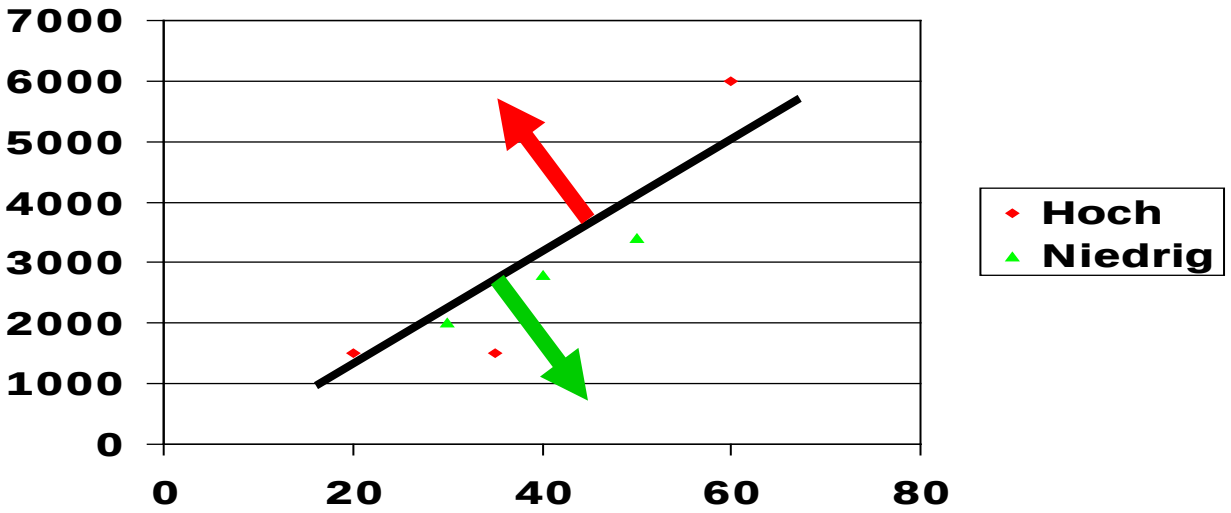
Beispiel



	Hoch	Niedrig
Hoch	2	0
Niedrig	1	3

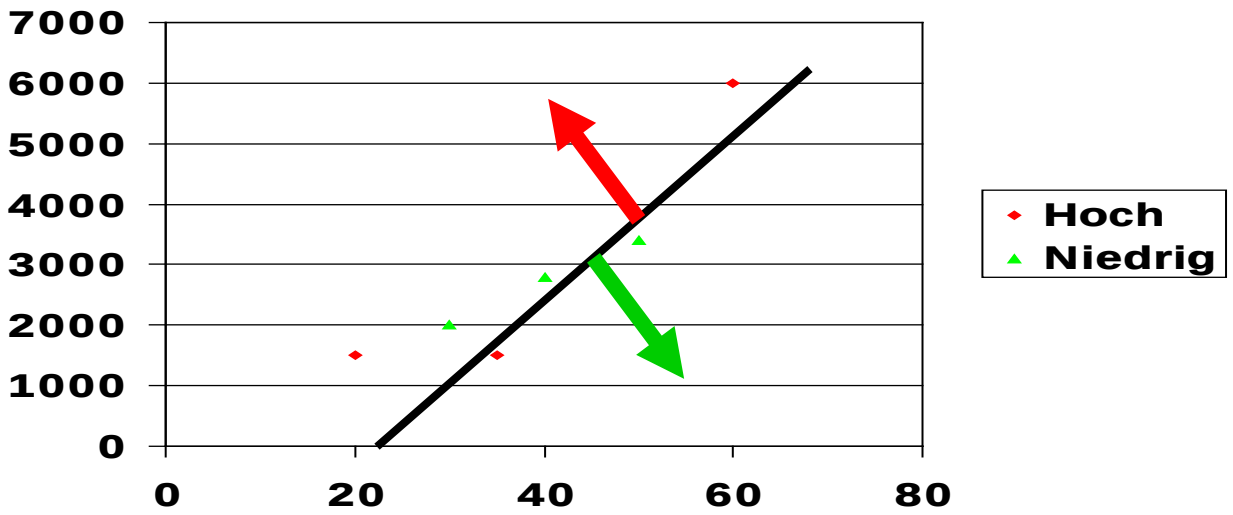
- Precision = TP/(TP+FP)= 2/2
  - Wenn wir hohes Risiko vorhersagen, haben wir immer Recht
- Recall = TP/(TP+FN) = 2/3
  - Aber wir verfehlen 33% der riskanten Kunden

Verbesserung



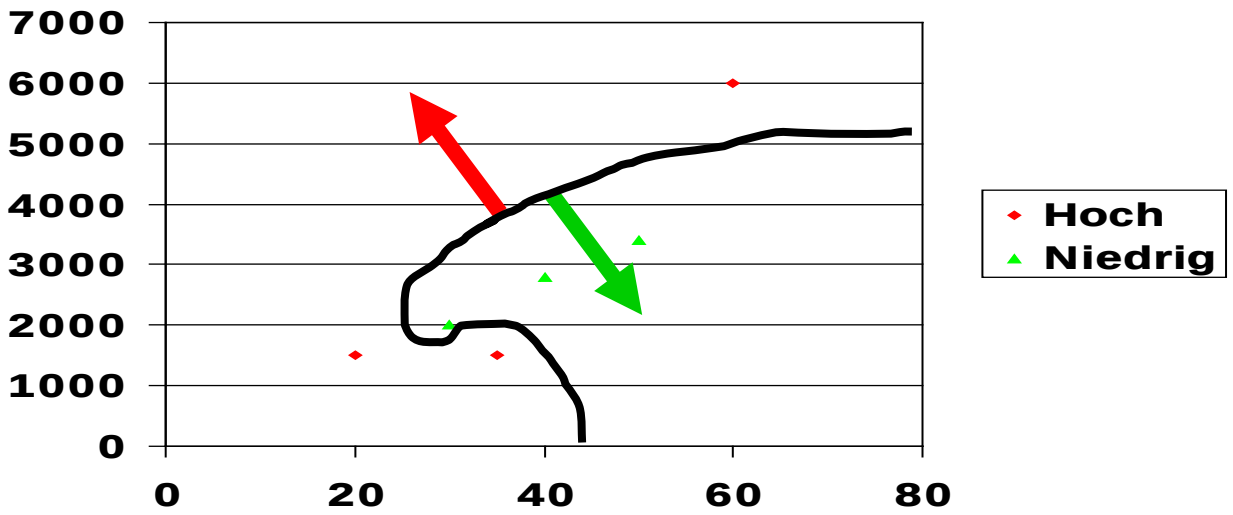
- Was können wir tun?

Verbesserung



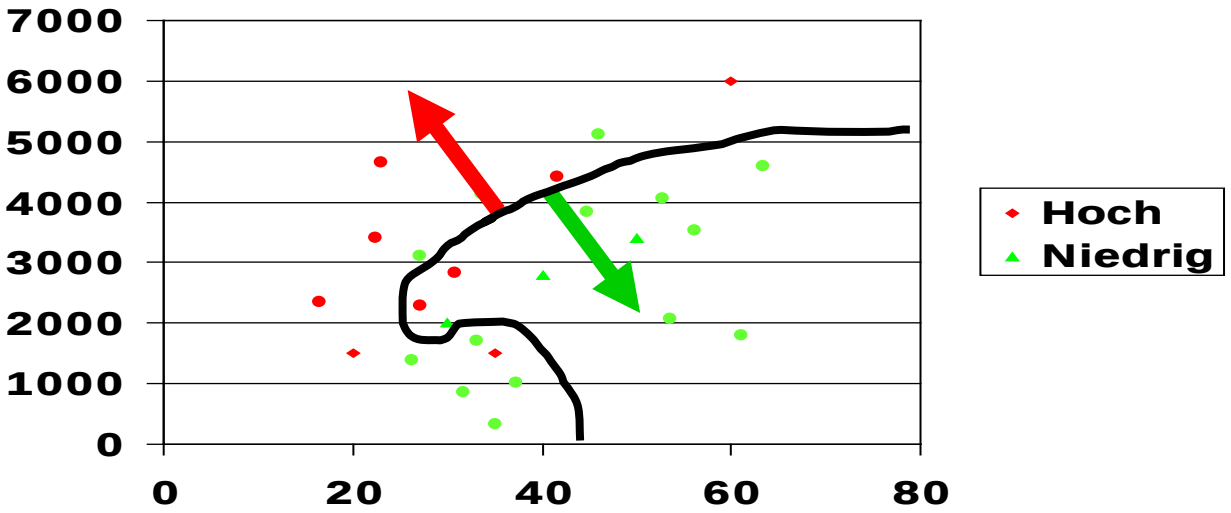
- Was können wir tun?
- Verschieben der Gerade
  - Nach unten: Erhöhter Recall, verringerte Precision
  - Nach oben: Verringerter Recall, erhöhte Precision
  - Die meisten Daten sind linear nicht vollständig trennbar

Verbesserung



- Was können wir tun?
- Nichtlineare Trennfunktionen betrachten
  - Lernen ist komplexer
  - Welche Klasse von Funktionen soll man nehmen?
  - Mit genügend Parametern kann man immer eine perfekte Funktion finden (accuracy = 1)

Overfitting



- Man lernt und evaluiert immer nur auf einem Ausschnitt der Realität
- Perfekte Trennfunktionen generalisieren idR nicht
- Oft schlechte Performanz auf neuen Daten
- Erfahrung: Lineare Funktionen sind gar nicht schlecht

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

Einschränkung

- (Lineare) Trennfunktionen kann man nur bestimmen, wenn die Werte aller Attribute geordnet sind und als Zahlen kodiert werden können
- Was tun bei kategorialen, ungeordneten Attributen?

Kategoriale 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

- Kategoriale Attribute kann man i.A. nicht als Parameter von Trennfunktionen benutzen
- Aber man könnte Regeln ableiten
  - 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 Regeln?
  - 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

- Overfitting
  - 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 = niedrig
    elseif Alter=43 and Autotype=Sport then Risikoklasse = hoch
    else Risikoklasse = niedrig

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

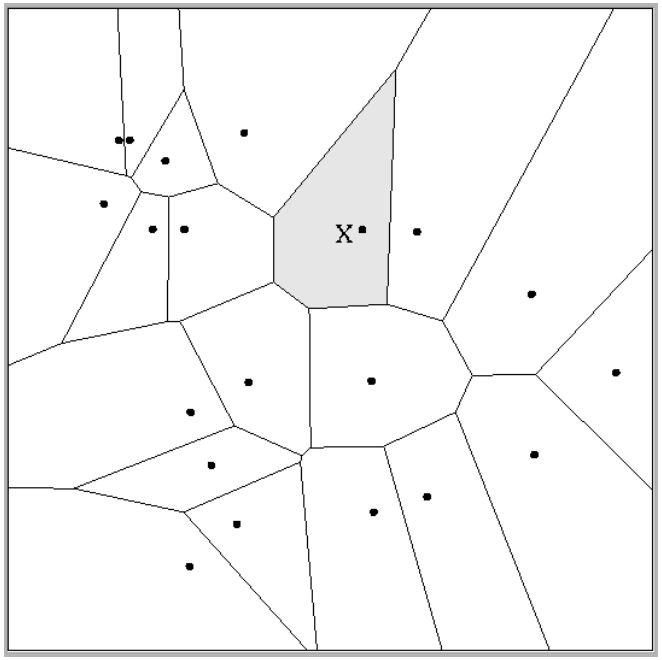
Inhalt dieser Vorlesung

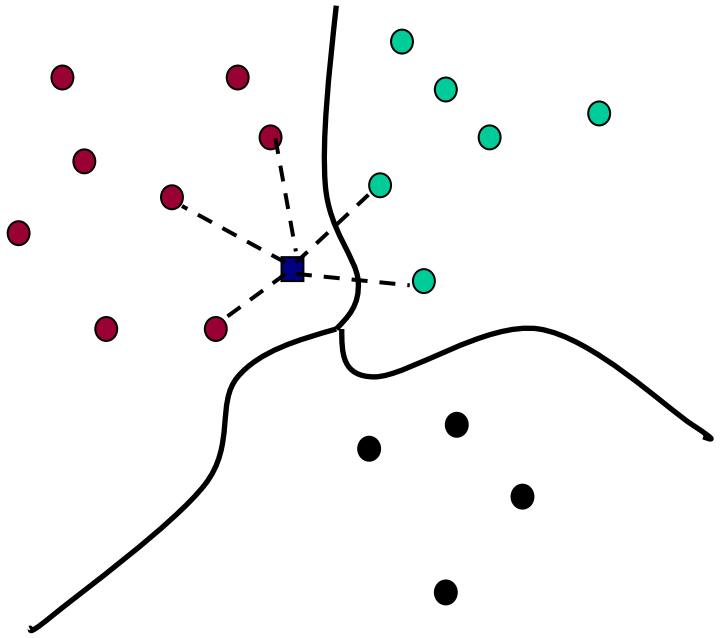
- Einführung
- kNN Klassifikator
- Naive-Bayes Klassifikator
- Entscheidungsbäume

Nearest Neighbor Klassifikator

- Definition
  Sei T die Trainingsmenge, d eine Abstandsfunktion und t ein zu klassifizierendes Objekt
  - *Ein nearest-neighbor (NN) Klassifikator weist t der Klasse des Objekts t'∈T zu, dass minimalen Abstand zu t hat.*
  - *Ein k-nearest-neighbor (kNN) Klassifikator weist t der Klasse zu, die am häufigsten unter den k zu t nächsten Objekten ist.*
- Bemerkungen
  - Sehr einfach, oft sehr effektiv, kNN meist besser als NN
  - „Lazy learner": Es gibt kein Modell (die Daten sind das Modell)
  - Sehr nahe zu Case-Based Reasoning
  - Schlecht bei wenigen oder stark clusternden Trainingsdaten
  - Variante: Klassen der k-nächste Objekte mit Abstand gewichtet

Illustration

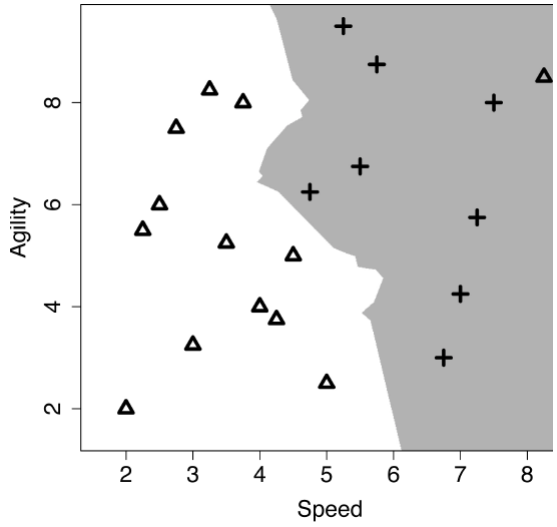




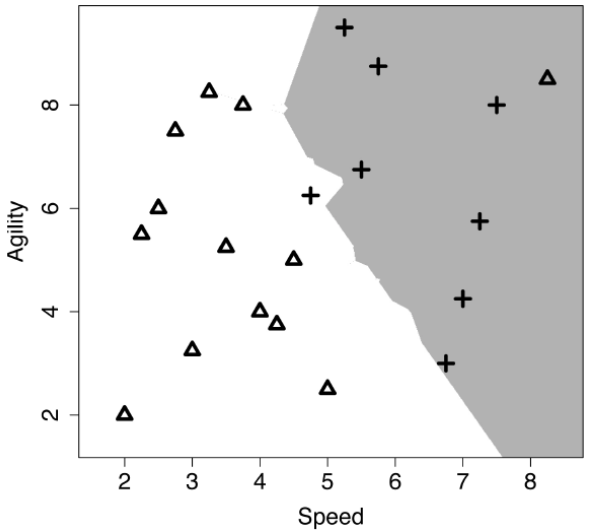
1NN und Voronoi diagram

5NN

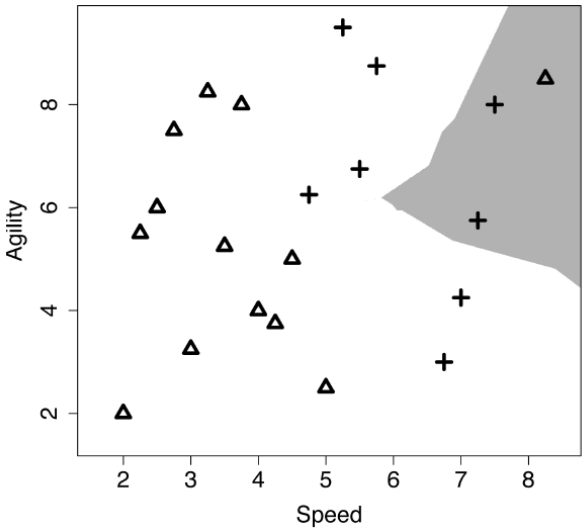
Welches k?



k=3



k=5



k=15

- Je größer k, desto homogener werden die Bereiche, desto weniger Übergänge gibt es, desto mehr wird die Majority-Klasse bevorzugt

Abstandsfunktion

- Qualität (und Laufzeit) hängt zentral von der Abstandsfunktion ab
  - Siehe auch Abstandsfunktion bei Clustering
- Beispiel: Euklidischer Abstand
  - Gleichgewichtung aller Attribute (warum?)
  - Kein Ausgleich von Unterschieden im Wertebereich
    - Alter: 18-80; Einkommen: 0-200.000
    - Lösung: min-max Normierung; Z-Scores
  - Keine Berücksichtigung der Verteilung von Werten
- Kategorialer Attribute brauchen spezielle Behandlung
  - Beispiel: Abstand 0 bei Gleichheit, sonst uniformer Abstand 1
- Abstandsfunktionen kann man lernen
  - Objekte der gleichen Klasse sollen nahe beieinander liegen und weit weg von anderen

Nachteil

- Problem: Geschwindigkeit
  - Man muss Abstand jedes Objekts aus T zu t berechnen
- Abhilfen
  - T in Indexstruktur packen, die Nachbarschaftssuche unterstützt
    - M-Tree, M-Index
  - Nicht ganz T nehmen, sondern ein zufälliges Sample
  - T erst clustern, dann einen Vertreter pro Cluster wählen
- Trade-Off: Letzten zwei Abhilfen führen zur Verschlechterung der Vorhersagequalität, aber Verbesserung der Laufzeit

Inhalt dieser Vorlesung

- Einführung
- kNN Klassifikator
- Naive-Bayes Klassifikator
- Entscheidungsbäume

Bayes‘sche Klassifikation

- Einfache, wahrscheinlichkeitsbasiertes Verfahren
- Geg.: Klassenmenge C, Trainingsmenge T
- Wir suchen für ein Objekt t die Wahrscheinlichkeiten $p(c_i|t)$ der Zugehörigkeit zu jeder Klasse $c_i \in C$
- t wird Klasse mit höchste Wahrscheinlichkeit zugewiesen
- Objekte werden durch Feature $F=\{f_1, ..., f_n\}$ beschrieben

$$p(c \mid t) = p(c \mid f_1[t],..., f_n[t]) = p(c \mid f_1,..., f_n)$$

  - Jeder Featurewert ist also ein Zufallsexperiment
  - Objekt = gleichzeitiges Eintreten vieler Zufallereignisse

Featurewerte

- Wir definieren im Folgenden ein Verfahren für binäre Feature – ein Objekt „hat“ ein Feature oder nicht
- Direkt anwendbar für kategoriale Attribute
  - Aus jedem Attributwert ein eigenes Feature machen
- Schwieriger: integer, real
  - Diskretisierung (Binning)
    - Betrachtung als kat. Attribut verliert aber Ordnungsinformation
  - Feature über Bereichen definieren
    - Feature = „A>30“
    - Problem: Welche Bereiche?
    - Explorative Datenanalyse: Verteilung von Werten über Klassen ansehen

Wahrscheinlichkeiten

- Gesucht: p(c|t)
- Wir haben
  - T mit binären Features $f_1,...,f_n$
  - 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 T 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 formen um und wenden Bayes‘ Theorem an

$$p(c \mid f_1,..., f_n) = \frac{p(f_1,..., f_n \mid c) * p(c)}{p(f_1,..., f_n)} \approx p(f_1,..., f_n \mid c) * p(c)$$

Naive Bayes

- Wir haben $p(c \mid t) \approx p(f_1, \ldots, f_n \mid c) * p(c)$
- Den ersten Term kann man bei vielen Feature (viele Bins, viele Dimensionen) nicht vernünftig lernen
  - Exponentiell viele Kombinationen
  - Jede einzelne Kombination ist in jedem realen T sehr selten
  - Zählen ergibt stark verzerrte Häufigkeiten
- „Naive“ Annahme: Statistische Unabhängigkeit
  - Dann gilt $p(f_1, \ldots, f_n \mid c) = p(f_1 \mid c) * \ldots * p(f_n \mid c)$
- Und damit $p(c \mid t) \approx p(c) * \prod_{i=1}^{n} p(f_i \mid c)$

Beispiel

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

- Wir „raten“ binäre Features
  - Jung (Alter<25)?
  - Sportwagen?
- Damit
  - P(c=hoch)=3/5
  - $p(f_1=jung)=2/5$
  - $P(f_2=sportwagen)=2/5$
  - P(jung|hoch)=2/3, p(jung|niedrig)=0
  - P(sportwagen|hoch)=2/3, p(sportwagen|niedrig)=0
  - Smoothing: p(jung|niedrig)= p(sportwagen|niedrig)=0.01
- Neuer Kunde, 24, LKW
  - p(hoch|jung,¬sportwagen)=p(hoch)*p(jung|hoch)*(1-p(sportwagen|hoch)=3/5*2/3*1/3 = 6/45
  - p(niedrig|jung, ¬sportwagen)=2/5*0.01*0.99
- Neuer Kunde, 35, Familienwagen
  - p(hoch|alt,¬sportwagen)=3/5*1/3*1/3 = 3/45
  - p(niedrig|alt, ¬sportwagen)=2/5*0.99*0.99

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), assign(oid, cid)`**
  - Werte müssen noch in relative Häufigkeiten umgerechnet werden

- Alle $p(c_j)$

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

- Alle $p(f_i)$

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

- Alle $p(f_i|c_j)$

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

Multinomial Naive Bayes

- Numerische Features verlangen bisher Diskretisierung, Smoothing, und Feature Engineering (wegen Ordnung)
- Besser: Multinomial Naive Bayes
  - Annahme: Attribut ist normalverteilt mit Mittelwert μ und Standardabweichung σ

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

  - Damit kann man die Wahrscheinlichkeit jedes Werts x schätzen
  - Diese anstelle von $p(f_i)$ bzw. $p(f_i|c)$ verwenden
    - Kein Zählen relativer Häufigkeiten mehr
  - Idee auch für andere Verteilungen anwendbar

Fazit

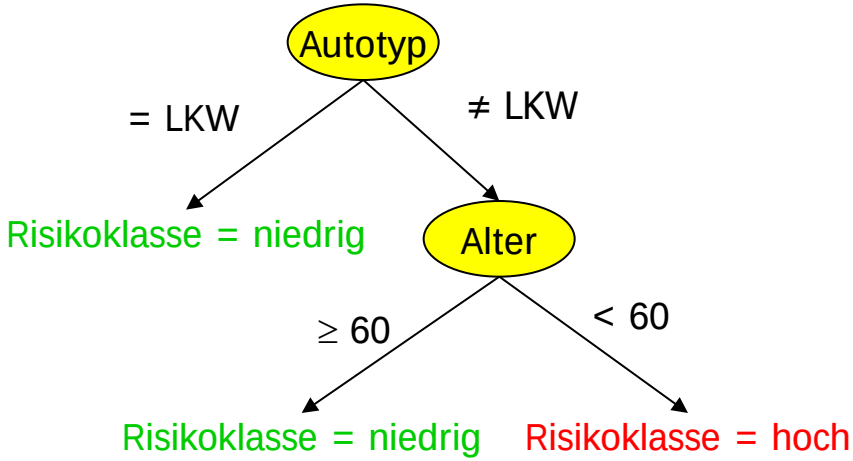
- Vorteile
  - Einfach, schnell
  - Modell kann inkrementell aktualisiert werden
  - Platzbedarf vernachlässigbar
    - Hängt nur von der Anzahl Features und Klassen ab
  - Oftmals schon gute Ergebnisse (typische Baseline)
- Nachteile
  - Geringe Erklärungskraft
  - Keine Beachtung korrelierter Feature
  - Feature Engineering ist komplex (Clustern)
  - Unabhängigkeitsannahme stimmt meistens nicht
    - Klappt oft trotzdem gut
    - Wenn nicht, stärkere Modelle nehmen (z.B. Bayes‘sche Netzwerke)

Inhalt dieser Vorlesung

- Einführung
- kNN Klassifikator
- Naive-Bayes Klassifikator
- Entscheidungsbäume
  - Grundprinzip
  - ID3 und CART
  - Tree Pruning

Entscheidungsbäume

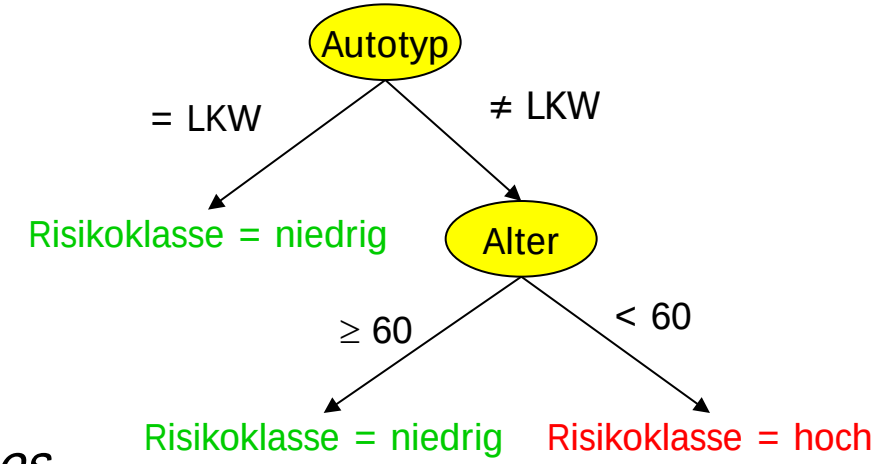
ID	Alter	Autotyp	Risiko
1	23	Familie	hoch
2	17	Sport	hoch
3	43	Sport	hoch
4	68	Familie	niedrig
5	32	LKW	niedrig



- Ableitung einer Menge von Regeln
- Hierarchische Anordnung: Bedingungen entlang eines Pfades werden ver-UNDet
- Blätter tragen Wahrscheinlichkeiten für Klassen
  - Weil weitere Auftrennung nicht möglich oder ...
  - um Overfitting zu verringern (später)

Formal

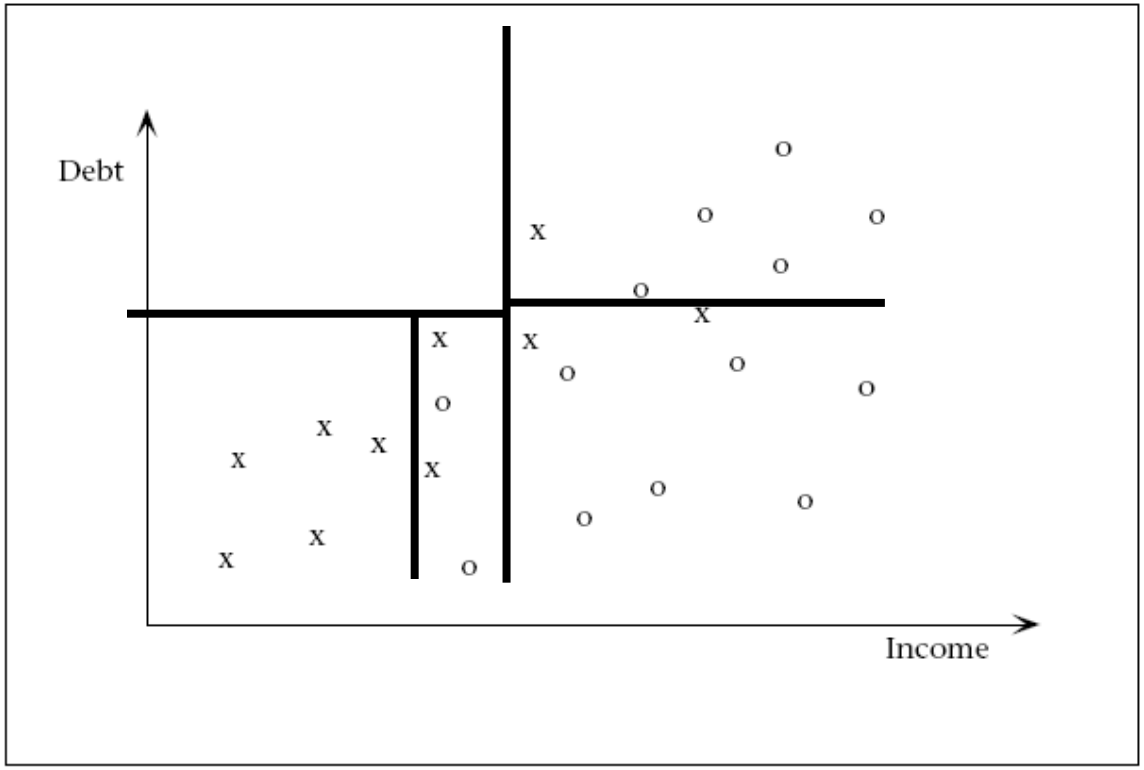
- Definition
  Ein Entscheidungsbaum ist ein Baum mit
  - *Jeder innere Knoten ist mit einem Attribut beschriftet*
  - *Die ausgehenden Kanten eines inneren Knoten sind mit Bedingungen an die Werte des Vaterknotens beschriftet, deren Wertebereiche den Wertebereich des Attributs überlappungsfrei abdecken*
  - *Die Konjunktion der Bedingungen eines Pfades ist immer erfüllbar*
  - *Blätter sind mit einer Wahrscheinlichkeitsverteilung über den Klassen beschriftet*



Anwendung und Berechnung

- Anwendung zur Klassifikation ist einfach
  - Gegeben neues Objekt t
  - Laufe den Baum ab und beschreite an jedem Knoten die ausgehende Kante, deren Bedingung für t zutrifft
  - Wird ein Blatt erreicht: Ordne t der Klasse zu, die in diesem Blatt die höchste Wahrscheinlichkeit hat
  - Laufzeit: O(Baumtiefe*log(|splits|)) – sehr schnell bei kleinen Bäumen und wenigen Splits
- Erstellung ist schwieriger
  - Ableitung eines „guten“ Entscheidungsbaums aus den Daten
    - Hohe Genauigkeit, gute Generalisierung, schnell
  - Decision Tree Induction (DTI)

Partitionierung



- DT‘s partitionieren den Featureraum rekursiv

DT: Pro und Contra

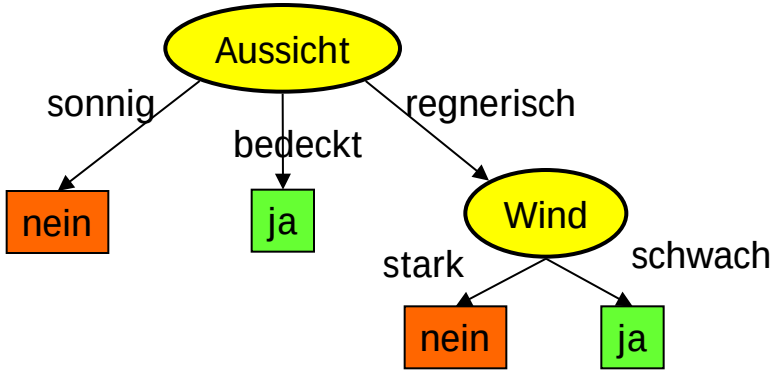
- Vorteile
  - Verständliche Darstellung – Anwender lieben DT
  - Leichte Ableitung von Regelmengen möglich
    - Einfache Implementierung
  - Schnelle Klassifikation (Baumhöhe typischerweise klein)
- Nachteile
  - DTI ist langsam
  - Typische DTI-Algorithmen berechnen nur rechtwinklige Partitionierung des Raums
    - Starke Einschränkung auf der Menge möglicher Trennfunktionen
  - Wahl der Split-Attribute in hoch dimensionalen Daten schwierig
    - Alle trennen gleich gut auf – Korrelationen werden nicht beachtet
  - Gefahr des Overfitting (siehe Tree Pruning)

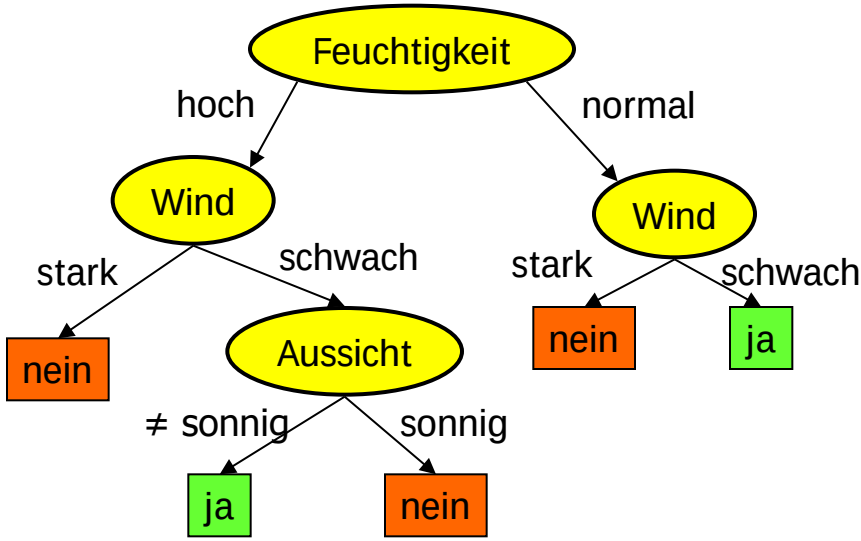
DTI: Grundverfahren

- Gegeben: Trainingsmenge T
  - Wähle ein Attribut A und Bedingungen (Splits) $B_1,...B_m$
    - 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 Fragen: Welches Attribut? Welche Splits? Wann stoppen?

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	. . .	. . .	. . .	. . .	. . .





Wie sieht ein guter DT aus?

- Prinzipiell: „klein“ ist gut
  - Schnelle Entscheidungen
  - Bessere Generalisierung, weniger Overfitting
- Mögliche konkrete Kriterien
  - Minimale Menge innerer Knoten
  - Minimale Baumtiefe (kürzester längster Ast)
  - Minimale Menge von Fehlern auf Trainingsmengen
    - Vorsicht: Overfitting
  - Finden optimaler Lösungen für alle diese Kriterien ist in NP
- Praxis: Heuristiken, die lokale Entscheidungen treffen
  - Reinheit der Partitionen in jedem Schritt maximieren

Einschub: Entropie

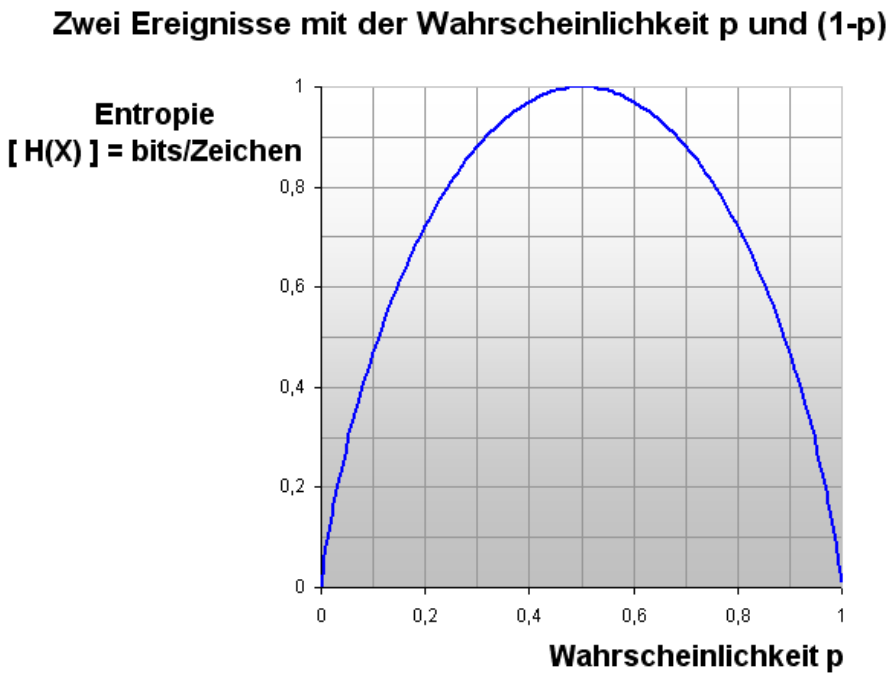
- Wie misst man die „Reinheit“ einer Partition?
- Definition
  Die Entropie einer Trainingsmenge T bzgl. Klassen $C=\{c_1,...c_n\}$ ist definiert als

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

  - *mit p_i = relative Häufigkeit der Klasse c_i in T*
- Bemerkung
  - entropy(T) = 0: homogene Menge (ein p_i=1, alle anderen =0)
  - entropy(T) >>0: Verwirrung
  - entropy(T) maximal 1 für zwei Klassen

Beispiel

- T=(h,h,h,h,l,l,l,l), entropy(T)=1
- T=(h,h,h,h,h,h,h,l), entropy(T)=0,54...



Quelle: Wikipedia

Klassische DTI Algorithmen [Qui86, Qui93]

- Viele Varianten: ID3 / CART / C4.5 / C5.0
- Ziel: Anzahl der erwarteten Vergleiche minimieren
  - Also Baum balanciert und niedrig halten
- Prinzipielle Heuristik: Wähle Splits so, dass die Entropie der neuen Partitionen möglichst klein ist
- Eingangspartition hat auch schon eine Entropie
- Verbesserung hängt davon ab, wie sich die Entropie beim Split verändert: Information Gain
- Dabei zählen grosse Partitionen mehr als kleine

Information Gain

- Definition

  Der Information Gain einer Partitionierung P (Split) einer Grundmenge T mit |P|=m ist definiert als

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

  Der gewichtete Information Gain ist definiert als

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

- Bemerkung
  - Triviale Lösung mit m=|T| will man natürlich verhindern
  - Prinzipiell ist der Information Gain je höher, je größer m ist
  - IdR fixiert man m=2

Beispiel

Klasse

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

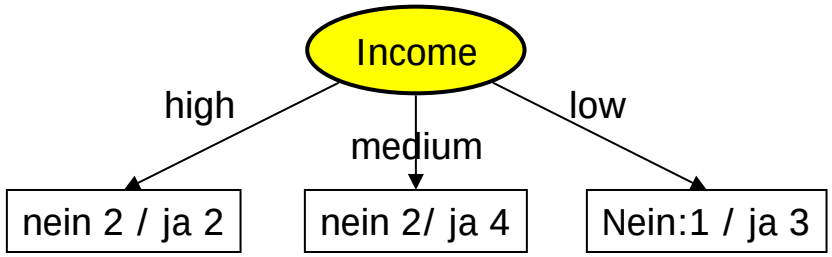
Quelle: [HK06]

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

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

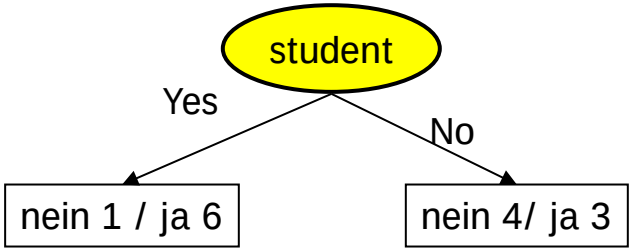
- Split bei income?
  - Drei Partitionen „high“, „medium“, „low“
  - entropy(high) = -2/4*log(2/4) – 2/4*log(2/4) ~ 1
  - entropy(medium) = -2/6*log(2/6) – 4/6*log(4/6) ~ 0.91
  - entropy(low) = -3/4*log(3/4) – ¼*log(1/4) ~ 0.81
  - wgain(income) = 0.94–(4/14*1+6/14*0.91 + 4/14*0.81) ~ 0.3



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

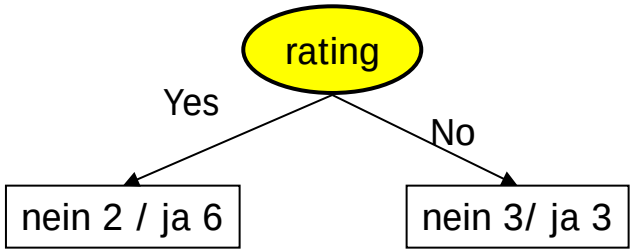
- Split bei student
  - Zwei Partitionen
  - entropy(yes) = -6/7*log(6/7) – 1/7*log(1/7) ~ 0.59
  - entropy(no) = -3/7*log(-3/7) – 4/7*log(4/7) ~ 0.98
  - wgain(student) = 0.94 – (7/14*0.59 + 7/14*0.98) ~ 0.15



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
  - entropy(fair) = -2/8*log(2/8) – 6/8*log(6/8) ~ 0.81
  - entropy(excellent) = -3/6*log(-3/6) – 3/6*log(3/6) ~ 1
  - wgain(rating) = 0.94 – (8/14*0.81 + 6/14*1) ~ 0.04



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() maximal ist
  - Partitioniere T gemäß A_i in Partitionen P_{ij}
  - Wende Verfahren rekursiv für jede neue Partition an, bei der Entropie!=0
- Probleme
  - Die Doppelschleife ist extrem teuer für Attribute mit vielen Werten (z.B. numerische Attribute)
  - Man muss theoretisch jede mögliche Aufteilung der Werte in beliebig viele Partitionen betrachten

Typischer Abhilfe: Nur binäre Splits

- Für geordnete Attribute: Alle Aufteilungen des Wertebereichs in zwei disjunkte Intervalle
  - Bei k verschiedenen Attributwerten sind das k-1 mögliche Splits
- Für andere Attribute: Alle Aufteilungen der Werte in zwei disjunkte Gruppen
  - Bei k Attributwerten sind das 2^k mögliche Splits
- Zusammen: Für jeden inneren Knoten
  - [Wenn z.B. alle Attribute numerisch sind]
  - Sei $j=\max(|A_i|)$, n Attribute, $|T|=m$
  - Alle Werte eines Attributs sortieren: $n*O(j*\log(j))$
  - Pro Attribut $O(j)$ mögliche Splits
  - Pro Split ganze Partition scannen, um wgain() auszurechnen
  - Zusammen: $O(n*j*\log(j) + n*j*m)$

Inhalt dieser Vorlesung

- Einführung
- Naive-Bayes Klassifikator
- Entscheidungsbäume
  - Grundprinzip
  - ID3 und CART
  - Tree Pruning
  - Varianten

Overfitting

- Trainingsmenge T, 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 T-Teilmengen
  - Entscheidungen haben bzgl. O nur noch geringe oder keine Aussagekraft mehr – zu wenig Beispiele vorhanden
- Abhilfe: Tree Pruning
  - Unterbäume abschneiden, die nur wenig zur Reduktion der Fehlerrate auf T beitragen

Tree Pruning

- Idee: Entferne (untere) Teilbäume, die
  - Nur noch kleine Partitionen haben (wenige Objekte betreffen)
  - Abschneiden erhöht die Gesamtfehlerrate auf T nur wenig
    - Und auf O hoffentlich gar nicht
- Einfache Methode: Tree Pruning mit „held back“ data
  - Teile T in Trainingsmenge T1 und Testmenge T2
  - Lerne Baum Tr1 auf T1
  - Entferne so lange Blattknoten von Tr1, wie der Fehler des Baums auf T2 kleiner wird
  - Problem: Benötigt viele Daten, Wahl von T1 / T2?
- Diverse andere Verfahren bekannt

Varianten

- Diverse Vorschläge, DTI effizienter zu implementieren
  - SPRINT, BOAT, ...
- Sehr kompetitiv: Random Forest
  - Berechne viele (e.g. 100) DT, aber jeden nur auf einem zufälligen Sample der Daten und zufälligem Sample der Feature
    - Implizite Feature Selection: Trees mit „guten“ Features tragen mehr zum Ergebnis bei
  - Evaluation immer auf den nicht gesampelten Trainingsdaten
    - Implizite Cross-Validierung
  - Klassifikation als gewichtetes Majority Voting
  - Sehr gutes und robustes Verfahren
- Weiterführung: X-Trees (XtremeTrees)
  - Wähle auch Attribute und Splits zufällig – viel schnelleres DTI
  - Benötigt mehr Bäume, aber schneller und gleich gut zu RF

Literatur

- Quinlan, J. (1993). "C4. 5: Programs for Machine Learning". San Francisco, Morgan Kaufmann.
- Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.
  - >30.000 Zitate
- Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. Machine learning, 63(1), 3-42.
- Han, J. and Kamber, M. (2006). "Data Mining. Concepts and Techniques", Morgan Kaufmann.
- Kelleher, Mac Namee, D'Arcy (2015): „Machine Learning for Predictive Data Analytics", MIT Press

Selbsttest

- Welche Klassifikationsverfahren haben wir kennengelernt?
- Was bedeutet Overfitting? Wie kann man es abschätzen?
- Welche Arten der Diskretisierung eines numerischen Attributs gibt es?
- Geg. folgende Daten eines binären Klassifikationsproblems. Berechnen Sie mit Naive Bayes $p(c_1|t)$ / $p(c_2|t)$
- Geben Sie die Komplexität der Entscheidungsfunktion für (a) k-NearestNeighbor, (b) Naive Bayes und (c) DTI mit Information Gain an