Klassifikation

Überblick

- **Grundkonzepte**
- Entscheidungsbäume
- Evaluierung von Klassifikatoren
- Lernen von Regeln
- Klassifikation mittels Assoziationsregeln
- Naïver Bayescher Klassifikator
- Naïve Bayes für Text Klassifikation
- Support Vektor Maschinen
- Ensemble-Methoden: Bagging und Boosting
- Zusammenfassung

Beispielanwendung

- Notfallaufnahme im Krankenhaus erfaßt 17 Variablen (z.B. Blutdruck, Alter, usw.) von neuen Patienten.
- Entscheidung ist notwenig: soll der Patient auf die Intensivstation verlegt werden.
- Patienten mit geringeren Überlebenschancen werden höhere Priorität eingeräumt.
- Problem: erkennen von Patienten mit hohem Risiko, Unterscheidung von Patienten mit geringem Risko.

Anderes Beispiel

- Kreditkartenfirma erhält tausende Anträge für neue Karten. Information auf dem Antrag sind,
  - Alter
  - Familienstand
  - Jährliches Gehalt
  - Schulden
  - Kredit-Rating
  - usw.
- Problem: soll der Antrag angenommen oder abgelehnt werden.

Maschinelles Lernen und unser Fokus

- Wie bei Menschen soll aus der Erfahrung aus der Vergangenheit gelernt werden.
- Computer haben keine “Erfahrung”.
- Ein Computer lernt aus Daten, welche “Erfahrungen” mit einer Anwendung repräsentieren.
- Unser Fokus: lerne eine Zielfunktion, die ein diskretes Klassenattribut vorhersagen kann.
- Diese Aufgabe wird Überwachtes Lernen oder induktives Lernen genannt.

Die Daten und das Ziel

- Daten: Eine Menge von Datensätzen (auch Beispiele, Instanzen oder Fälle), die wie folgt repräsentiert werden
  - *k* Attribute: $A_1, A_2, \ldots A_k$.
  - Eine Klasse: jedes Beispiel ist einer vorgegebenen Klasse zugeordnet.
- Ziel: lerne ein Klassifikationsmodell aus den Daten, das die Klasse von neuen, noch nicht klassifizierten Beispielen vorhersagen kann.

Beispiel: Daten (Kreditantrag)

Angenommen oder nicht

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	**No**
2	young	false	false	good	**No**
3	young	true	false	good	**Yes**
4	young	true	true	fair	**Yes**
5	young	false	false	fair	**No**
6	middle	false	false	fair	**No**
7	middle	false	false	good	**No**
8	middle	true	true	good	**Yes**
9	middle	false	true	excellent	**Yes**
10	middle	false	true	excellent	**Yes**
11	old	false	true	excellent	**Yes**
12	old	false	true	good	**Yes**
13	old	true	false	good	**Yes**
14	old	true	false	excellent	**Yes**
15	old	false	false	fair	**No**

Beispiel: Aufgabe

- Lerne ein Klassifikationsmodell aus den Daten
- Nutze das Modell um neue Anträge zu klassifizieren
  - Ja (angenommen) und
  - Nein (abgelehnt)
- Welche Klasse hat das folgende Beispiel?

Age	Has_Job	Own_house	Credit-Rating	Class
young	false	false	good	?

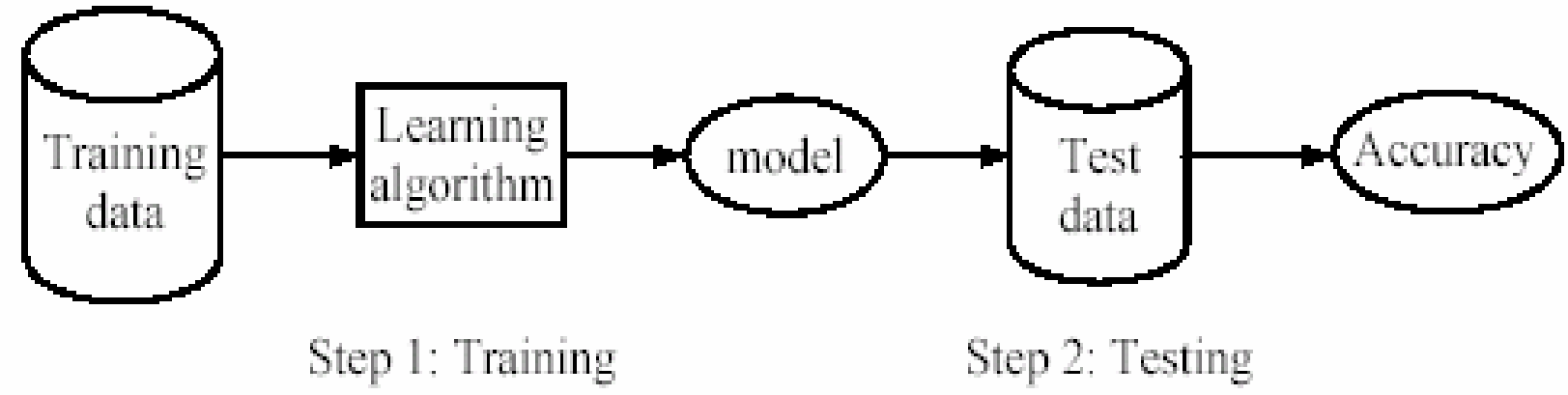
Überwachtes und Nicht-Überwachtes Lernen

- Überwachtes Lernen: Klassifikation ist überwachtes Lernen aus Beispielen.
  - Überwachung: Die Daten (Beobachtungen, Attribute, etc.) sind mit gegebenen Klassen verknüpft. Das ist als ob ein Lehrer die Klassen sagt (Supervision).
  - Testdaten sind auch schon vorklassifiziert.
- Unüberwachtes Lernen (Clustering)
  - Klassenzuordnungen der Daten sind unbekannt
  - Für eine gegebene Datenmenge sollen erst mal Klassen definiert werden

Überwachtes Lernen: Zwei Schritte

- Lernen (Training): Lerne ein Modell aus den Trainingsdaten
- Testen: Teste das Modell durch bisher nicht genutzte Daten um die Vorhersagegenauigkeit zu testen

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}},$$



Grundannahme des Lernens

Annahme: Die Verteilung der Trainingsbeispiele ist gleich der Verteilung der Testfälle (einschließlich alle zukünftigen Fälle).

- In der Praxis wird diese Annahme zu einem gewissen Grad immer verletzt.
- Starke Abweichungen ergeben geringe Klassifikationsgenauigkeit.
- Um gute Genauigkeit zu auf den Testdaten zu erreichen, müssen die Trainingsdaten hinreichend repräsentativ für die Testdaten sein.

Überblick

- Grundkonzepte
- Entscheidungsbäume
- Evaluierung von Klassifikatoren
- Lernen von Regeln
- Klassifikation mittels Assoziationsregeln
- Naïver Bayescher Klassifikator
- Naïve Bayes für Text Klassifikation
- Support Vektor Maschinen
- Ensemble-Methoden: Bagging und Boosting
- Zusammenfassung

Einführung

- Entscheidungsbaumlernen ist eine häufig eingesetzte Methode.
  - Klassifikationsgenauigkeit ist vergleichbar mit anderen Methoden und
  - es ist sehr effizient.
- Das Klassifikationsmodell ist eine Baum
- C4.5 von Ross Quinlan ist ein sehr bekanntes System.

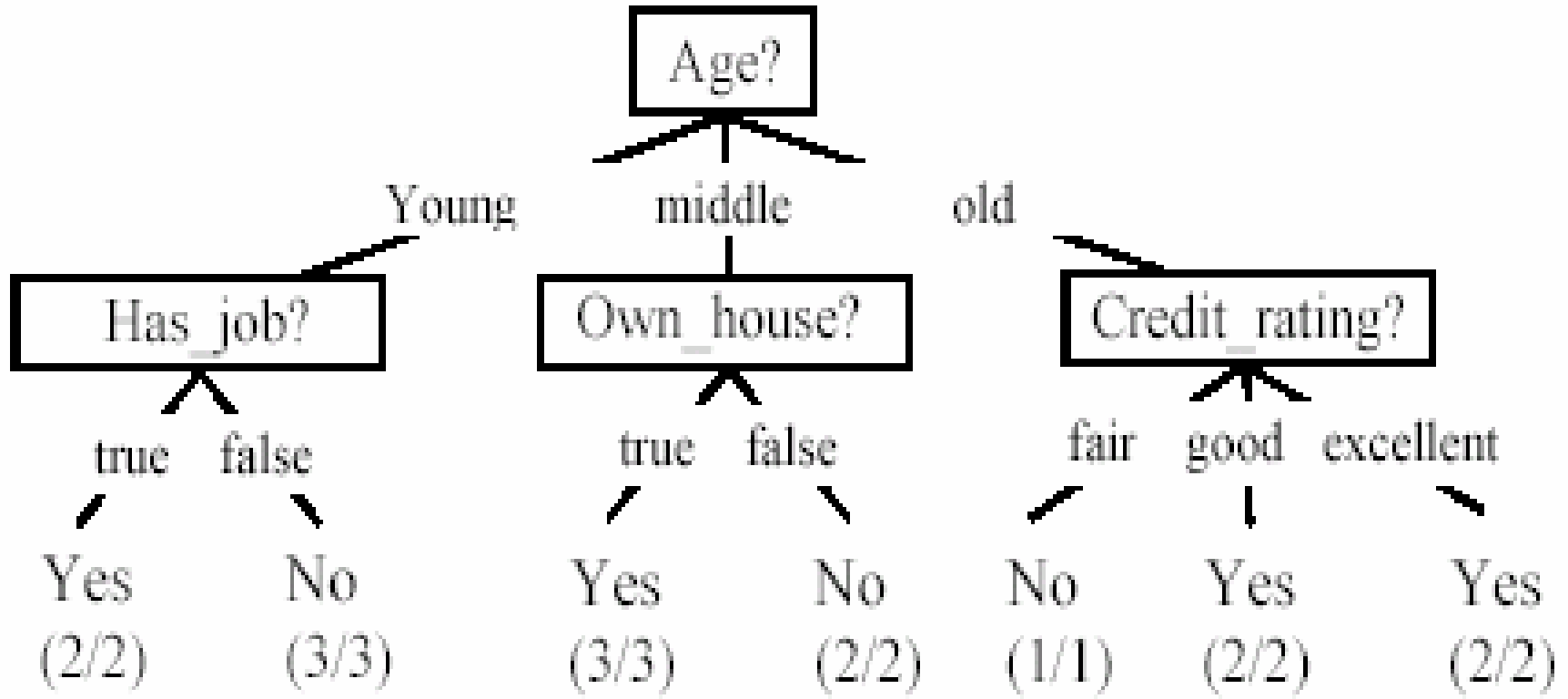
Beispiel: Daten (Kreditantrag)

Angenommen oder nicht

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	**No**
2	young	false	false	good	**No**
3	young	true	false	good	**Yes**
4	young	true	true	fair	**Yes**
5	young	false	false	fair	**No**
6	middle	false	false	fair	**No**
7	middle	false	false	good	**No**
8	middle	true	true	good	**Yes**
9	middle	false	true	excellent	**Yes**
10	middle	false	true	excellent	**Yes**
11	old	false	true	excellent	**Yes**
12	old	false	true	good	**Yes**
13	old	true	false	good	**Yes**
14	old	true	false	excellent	**Yes**
15	old	false	false	fair	**No**

Entscheidungsbaum für Kreditdaten

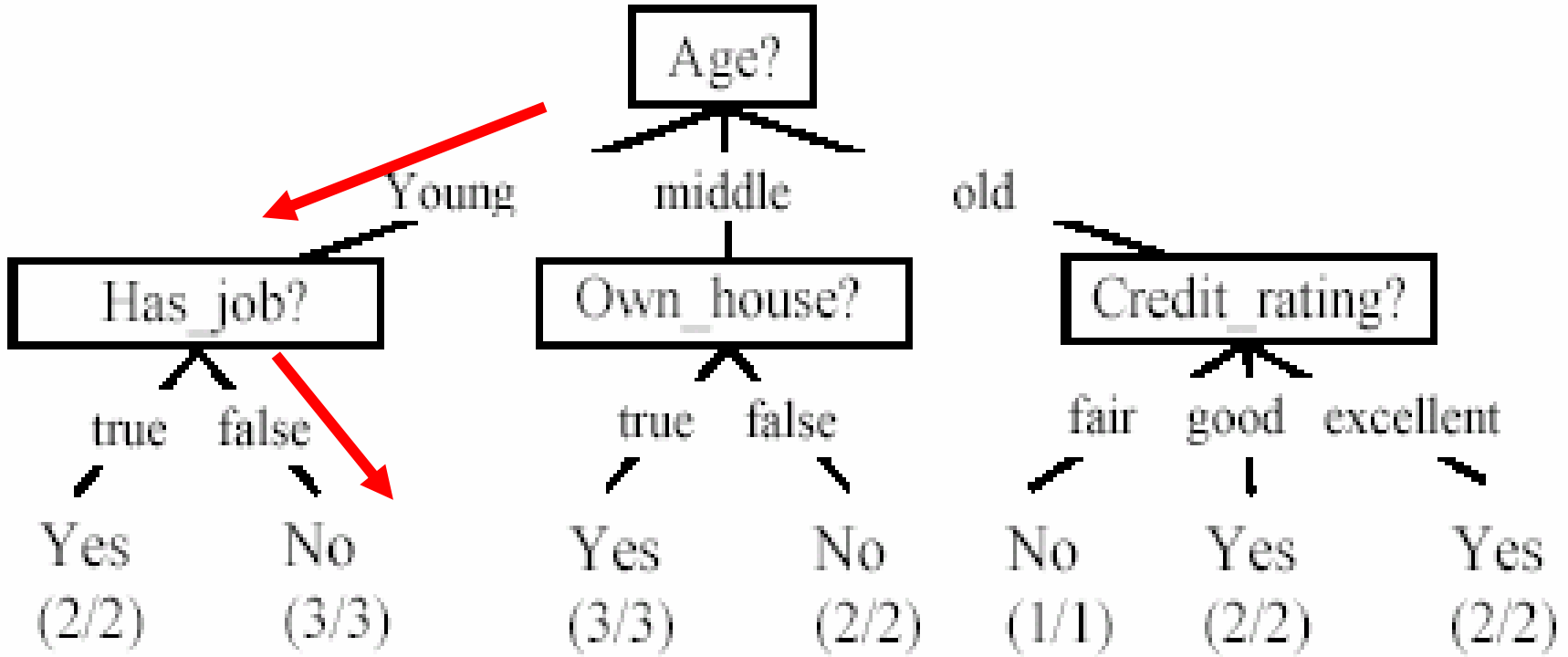
- Entscheidungsknoten und Blattknoten (Klassen)



Anwendung des Entscheidungsbaumes

Age	Has_Job	Own_house	Credit-Rating	Class
young	false	false	good	?

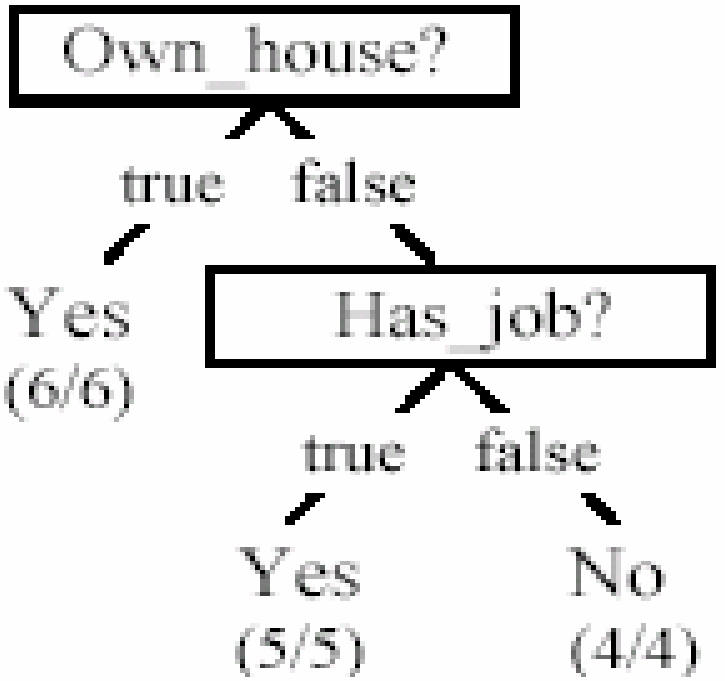
No



Ist der Entscheidungsbaum eindeutig?

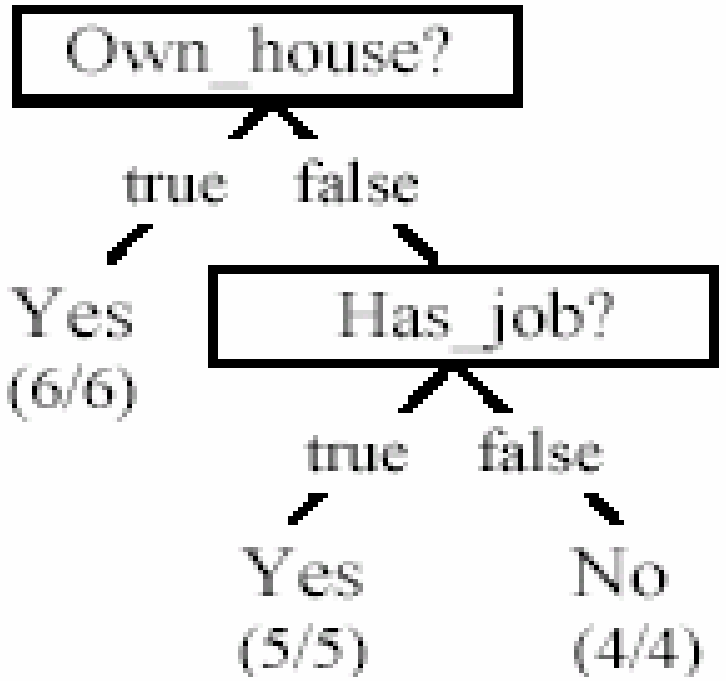
- Nein. Bei dem Beispiel gibt es noch einen einfacheren Baum.
- Kleine, genaue Bäume sind meist besser.
  - Philosophische Grundlage, William von Ockam.

- Den besten Baum zu finden ist NP-hard.
- Alle bisherigen Baumerzeugung Algorithmen sind Heuristiken



Von Entscheidungsbäumen zu Regelmengen

- Ein Entscheidungs-baum kann in eine Regelmenge kon-vertiert werden
- Jeder Pfad von der Wurzel zu einem Blatt ist eine Regel



Own_house = true → Class =Yes [sup=6/15, conf=6/6]
Own_house = false, Has_job = true → Class = Yes [sup=5/15, conf=5/5]
Own_house = false, Has_job = false → Class = No [sup=4/15, conf=4/4]

Algorithmus

- Grundalgorithmus (Greedy **divide-and-conquer** Algorithmus)
  - vorläufige Annahme, alle Attribute sind kategorisch (kontinuierliche Attribute können auch behandelt werden)
  - Baum wird top-down rekursiv konstruiert
  - Zu Beginn sind alle Trainingsbeispiele in der Wurzel
  - Beispiel werden rekursiv bezüglich der gewählten Attribute aufgeteilt.
  - Attribute werden auf der Basis einer Reinheitsfunktion gewählt (z.B., Information Gain)
- Abbruchbedingungen für das Aufteilen
  - Alle Beispiel in einem Knoten gehören zu der gleichen Klasse
  - Es gibt keine noch nicht gewählten Attribute – Mehrheitsklasse wird dann dem Blatt zugewiesen
  - Knoten hat zu wenig Bespiele

Algorithmus

. **Algorithm** decisionTree(D, A, T)

```
1     if D contains only training examples of the same class c_j ∈ C then
2         make T a leaf node labeled with class c_j;
3     elseif A = ∅ then
4         make T a leaf node labeled with c_j, which is the most frequent class in D
5     else  // D contains examples belonging to a mixture of classes. We select a single
6           // attribute to partition D into subsets so that each subset is purer
7         p_0 = impurityEval-1(D);
8         for each attribute A_i ∈ {A_1, A_2, ..., A_k} do
9             p_i = impurityEval-2(A_i, D)
10        end
11        Select A_g ∈ {A_1, A_2, ..., A_k} that gives the biggest impurity reduction,
              computed using p_0 − p_i;
12        if p_0 − p_g < threshold then     // A_g does not significantly reduce impurity p_0
13          make T a leaf node labeled with c_j, the most frequent class in D.
14        else                              // A_g is able to reduce impurity p_0
15            Make T a decision node on A_g;
16            Let the possible values of A_g be v_1, v_2, ..., v_m. Partition D into m
                  disjoint subsets D_1, D_2, ..., D_m based on the m values of A_g.
17            for each D_j in {D_1, D_2, ..., D_m} do
18                if D_j ≠ ∅ then
19                  create a branch (edge) node T_j for v_j as a child node of T;
20                  decisionTree(D_j, A-{A_g}, T_j)// A_g is removed
21                end
22            end
23        end
24    end
```

Auswahl des Attributes zum Aufteilen der Daten

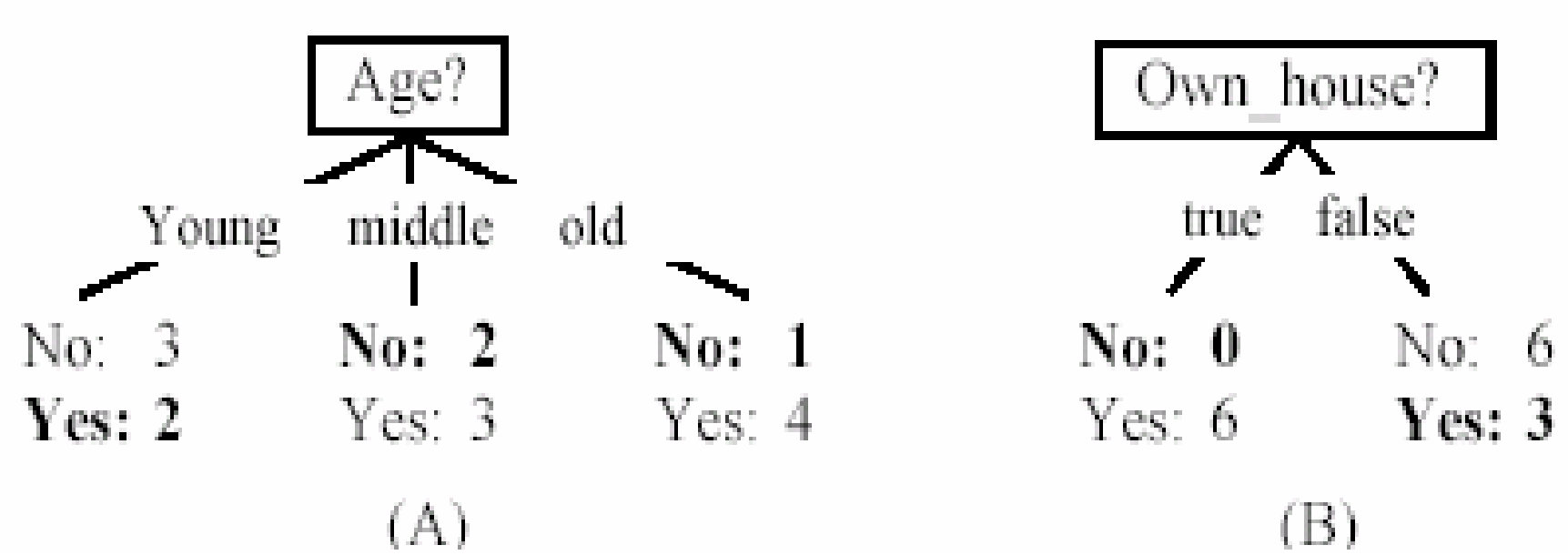
- Wichtige Entscheidung beim Aufbau eines Entscheidungsbaumes – welches Attribut wird zum Aufteilen gewählt
- Das Ziel ist die Unreinheit oder Unsicherheit in den Daten soweit wie möglich zu reduzieren
  - Ein Teilmenge der Daten ist rein, wenn alle Instanzen zur gleichen Klasse gehören
- Die *Heuristik* in C4.5 ist, das Attribut mit dem maximalen Information Gain oder Gain Ratio zu wählen

Beispiel: Daten (Kreditantrag)

Angenommen oder nicht

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	**No**
2	young	false	false	good	**No**
3	young	true	false	good	**Yes**
4	young	true	true	fair	**Yes**
5	young	false	false	fair	**No**
6	middle	false	false	fair	**No**
7	middle	false	false	good	**No**
8	middle	true	true	good	**Yes**
9	middle	false	true	excellent	**Yes**
10	middle	false	true	excellent	**Yes**
11	old	false	true	excellent	**Yes**
12	old	false	true	good	**Yes**
13	old	true	false	good	**Yes**
14	old	true	false	excellent	**Yes**
15	old	false	false	fair	**No**

Zwei mögliche Wurzeln, welche ist besser?



- (B) scheint besser zu sein.

Informationstheorie

- Informationstheorie liefert die mathematische Grundlage zum Messen von Informationsgehalt.
- Beispiel: Wieviel ist die Information über das Ergebnis eines Münzwurfs wert?
  - Wenn man schon eine gute Schätzung der Wahrscheinlichkeit der Münze hat, ist die Information über das tatsächliche Ergebniss weniger wert.
  - Wenn man zu Beispiel weiß, dass eine Münze so beeinflußt ist, dass mit Wahrscheinlichkeit 0.99 Kopf kommt, ist eine Antwort über den tatsächlichen Ausgang weniger wert, als wenn es eine faire Münze wäre.

Informationstheorie: Entropie

- Entropie

$$entropy(D) = -\sum_{j=1}^{|C|} \Pr(c_j) \log_2 \Pr(c_j)$$

$$\sum_{j=1}^{|C|} \Pr(c_j) = 1,$$

- Pr(c_j) ist die Wahrscheinlichkeit der Klasse c_j in der Datenmenge D
- Entropie ist ein Maß der Unreinheit oder Unordnung im den Daten

Entropie

1. The data set D has 50% positive examples ($\Pr(positive) = 0.5$) and 50% negative examples ($\Pr(negative) = 0.5$).

$$entropy(D) = -0.5 \times \log_2 0.5 - 0.5 \times \log_2 0.5 = 1$$

2. The data set D has 20% positive examples ($\Pr(positive) = 0.2$) and 80% negative examples ($\Pr(negative) = 0.8$).

$$entropy(D) = -0.2 \times \log_2 0.2 - 0.8 \times \log_2 0.8 = 0.722$$

3. The data set D has 100% positive examples ($\Pr(positive) = 1$) and no negative examples, ($\Pr(negative) = 0$).

$$entropy(D) = -1 \times \log_2 1 - 0 \times \log_2 0 = 0$$

- Wenn die Daten reiner sind, ist die Entropie kleiner.

Information gain

- Gegeben sei eine Menge von Daten *D*, berechne erst die Entropie:

$$entropy(D) = -\sum_{j=1}^{|C|} \Pr(c_j) \log_2 \Pr(c_j)$$

- Wenn Attribut A_i, mit v Ausprägungen die Wurzel des Baumes ist, wird *D* in v Teilmengen D_1, D_2 …, D_v partitioniert. Die erwartete Entropie wenn A_i Wurzel ist:

$$entropy_{A_i}(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times entropy(D_j)$$

Information gain

- Der Informationsgewinn durch Auswahl von Attribut A_i zum Aufteilen der Daten ist

$$gain(D, A_i) = entropy(D) - entropy_{A_i}(D)$$

- Das Attribut mit dem größten Informationsgewinn wird als Wurzel des aktuellen Baumes gewählt.

Beispiel

$$entropy(D) = -\frac{6}{15} \times \log_2 \frac{6}{15} - \frac{9}{15} \times \log_2 \frac{9}{15} = 0.971$$

$$entropy_{Own\_house}(D) = -\frac{6}{15} \times entropy(D_1) - \frac{9}{15} \times entropy(D_2)$$
$$= \frac{6}{15} \times 0 + \frac{9}{15} \times 0.918$$
$$= 0.551$$

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	excellent	No
3	young	true	false	good	Yes
4	young	true	true	good	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

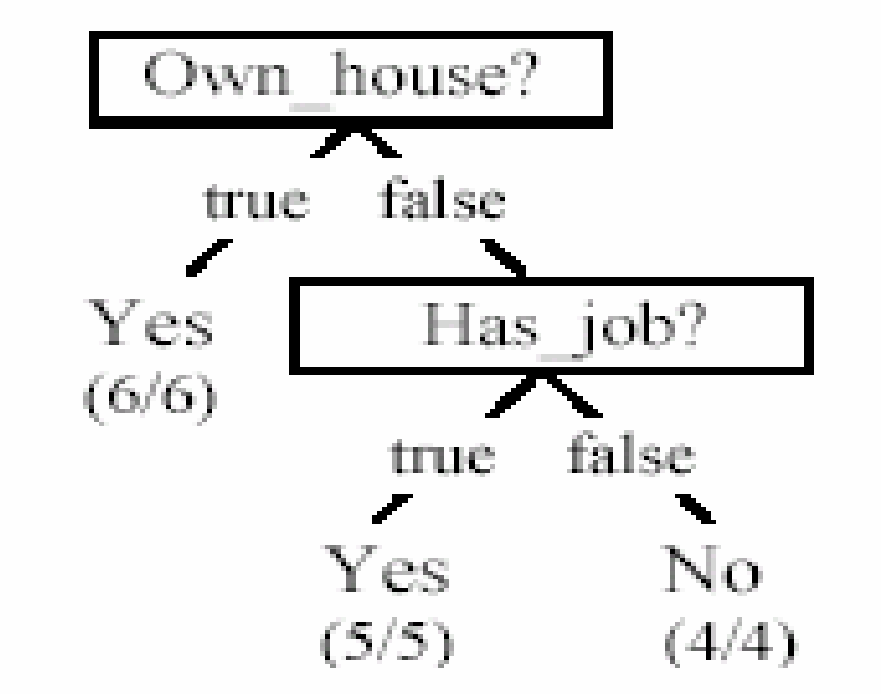
$$entropy_{Age}(D) = -\frac{5}{15} \times entropy(D_1) - \frac{5}{15} \times entropy(D_2) - \frac{5}{15} \times entropy(D_3)$$
$$= \frac{5}{15} \times 0.971 + \frac{5}{15} \times 0.971 + \frac{5}{15} \times 0.722$$
$$= 0.888$$

Age	Yes	No	entropy(Di)
young	2	3	0.971
middle	3	2	0.971
old	4	1	0.722

- Own_house ist die beste Wahl als Wurzel

$$gain(D, \text{Age}) = 0.971 - 0.888 = 0.083$$
$$gain(D, \text{Own\_house}) = 0.971 - 0.551 = 0.420$$
$$gain(D, \text{Has\_Job}) = 0.971 - 0.647 = 0.324$$
$$gain(D, \text{Credit\_Rating}) = 0.971 - 0.608 = 0.363$$

Erzeugter Baum

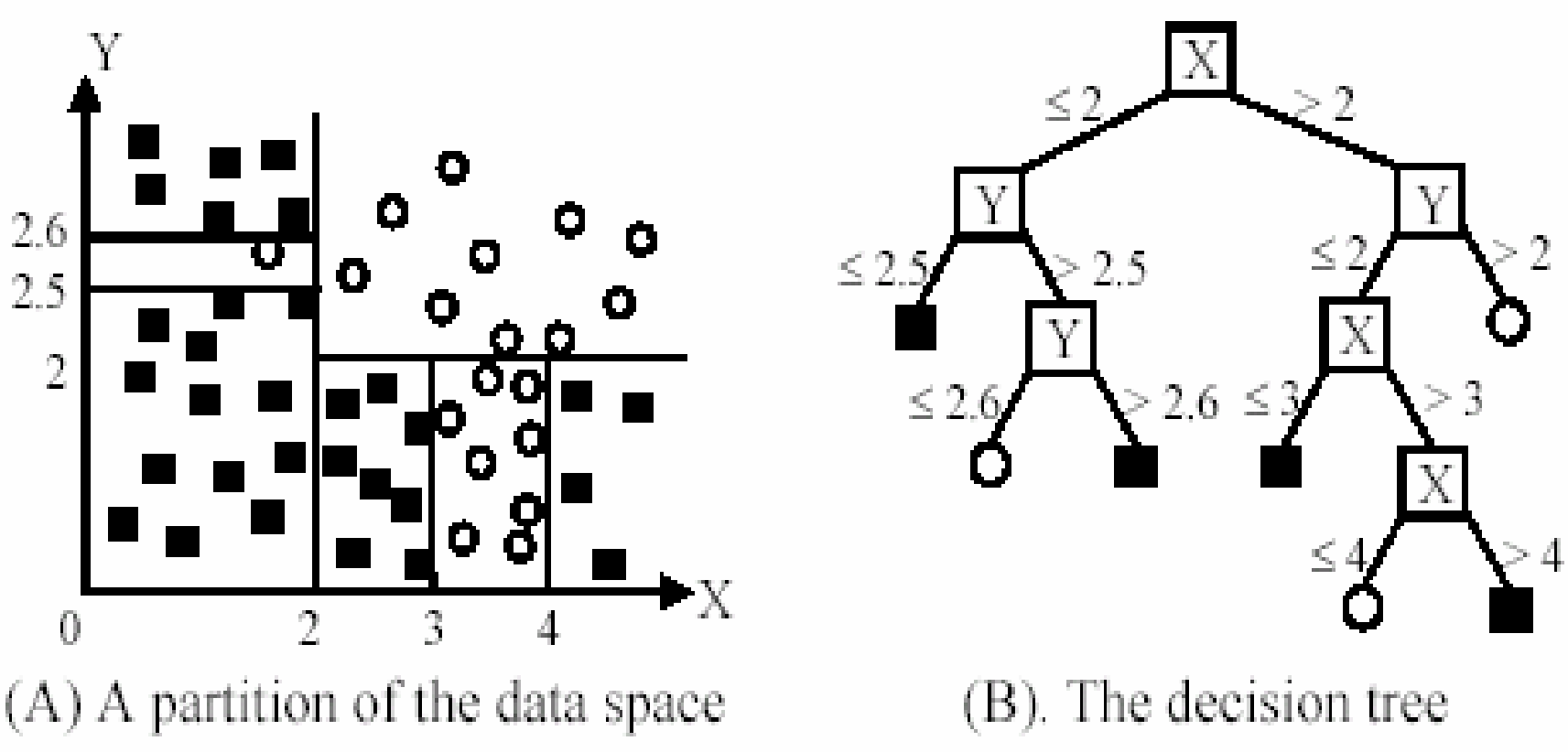


- Anstelle von Information gain kann auch der Gain ratio zum Messen der Unreinheit genutzt werden

Kontinuierliche Attribute

- Kontinuierliche Attribute werden in zwei Intervalle aufgeteilt
- Wie findet man die beste Genze?
  - Nutze Information Gain oder Gain ratio
  - Sortiere alle Werte des kontinuierlichen Attributes in aufsteigender Ordnung $\{v_1, v_2, \ldots, v_r\}$,
  - Die Grenzen zwischen den Werten v_i und v_{i+1} wirken sich gleich aus. Alle möglichen Grenzen werden getestet und die beste wird genommen.

Beispiel im kontinuierlichen Raum



(A) A partition of the data space

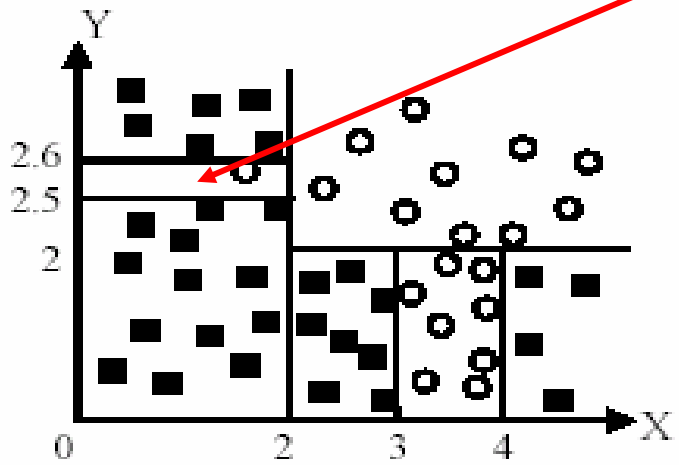
(B). The decision tree

Vermeide Overfitting bei der Klassifikation

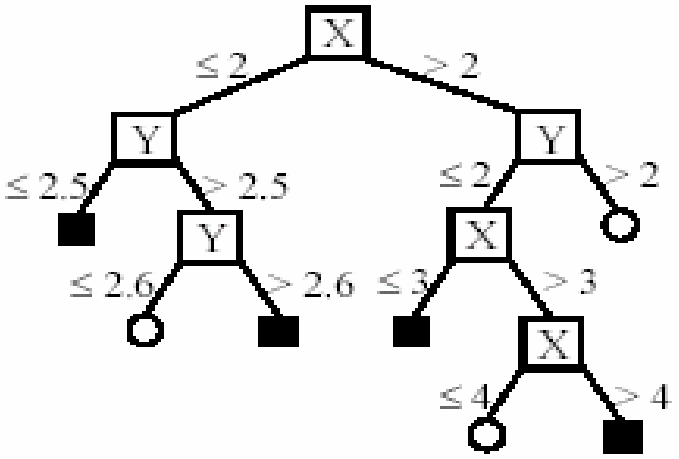
- Overfitting: Ein Baum kann zu stark an die Trainingsdaten angepasst sein
  - hohe Genauigkeit auf den Trainingsdaten aber schlecht auf den Testdaten
  - Symptome: Baum ist sehr hoch und zuviele Verzweiungen, manche spiegeln Anomalien wie Aussreißer und Rauschen wieder
- Zwei Ansätze und Overfitting zu vermeiden
  - Pre-pruning: breche die Baumkonstruktion früh ab.
    - Schwer zu entscheiden, da man nicht weiß, wie der Aufbau weitergeht.
  - Post-pruning: Schneide Zweige oder Teilbäume von dem "ausgewachsenen" Baum ab.
    - Oft genutzt, C4.5 nutzt statistische Methoden um Fehler an einem Knoten zu schätzen.
    - Validierungsmenge kann auch genutzt werden

Beispiel

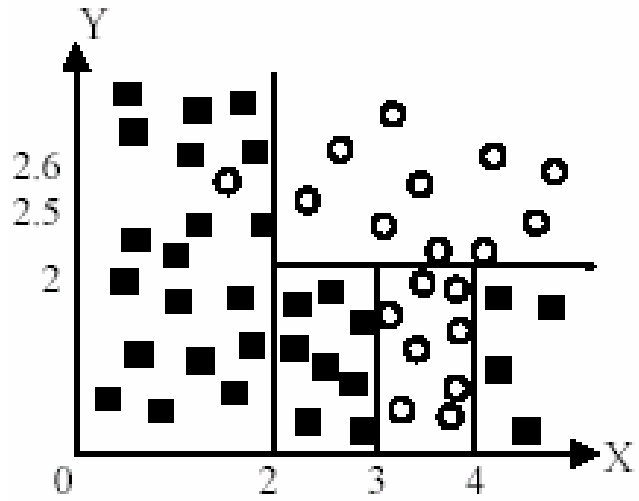
Wahrscheinlich Overfitting

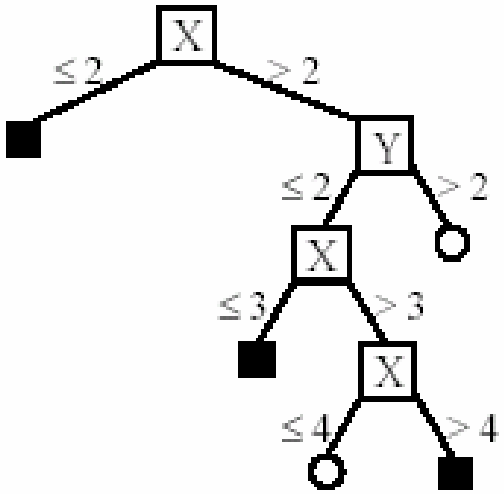


(A) A partition of the data space



(B). The decision tree





Überblick

- Grundkonzepte
- Entscheidungsbäume
- Evaluierung von Klassifikatoren
- Lernen von Regeln
- Klassifikation mittels Assoziationsregeln
- Naïver Bayescher Klassifikator
- Naïve Bayes für Text Klassifikation
- Support Vektor Maschinen
- Ensemble-Methoden: Bagging und Boosting
- Zusammenfassung

Evaluierung von Klassifikatoren

- **Vorhersagegenauigkeit**

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}}$$

- Effizienz
  - Zeit um das Modell zu konstruieren
  - Zeit um das Modell anzuwenden
- Robust: Behandlung von Rauschen und Ausreißern und fehlenden Werten
- Skalierbarkeit: Effizienz bei Nutzung von Festplattenspeicher
- Interpretierbarkeit:
  - Verstehbarkeit und Einsichten des Modelles
- Kompaktheit des Modells: Größe des Baumes, Anzahl der Regeln, ...

Evaluierungsmethoden

- **Zurückhalten von Daten**: Die verfügbaren Daten *D* werden in zwei disjunkte Teilmengen geteilt
  - *Trainingsdaten* D_{train} (zum Lernen des Modells)
  - *Testdaten* D_{test} (zum Testen des Modells)
- **Wichtig:** Trainingsdaten sollen nicht zum Testen genutzt werden und Testdaten nicht zum Trainieren
  - Bisher nicht genutzte Testdaten sind ein unbeeinflußter Test.
- Diese Methode funktioniert nur wenn *D* groß ist.

Evaluierungsmethoden

- **n-fache Kreuz-Validierung**: Die Daten werden in *n* gleichgroße disjunkte Teilmengen partitioniert.
- Nutze jede Teilmenge mal als Testmenge und trainiere auf der Vereinigung der restlichen *n*-1 Teilmengen
- Diese Prozedur wird *n* mal wiederholt, so erhält man *n* Genauigkeiten.
- So erhält man eine durchschnittliche Genauigkeit und eine Varianz der Genauigkeit.
- 10-fache oder 5-fache Kreuz-Validierung ist Standard
- Diese Methode funktioniert auch auf weniger großen Daten

Evaluierungsmethoden

- **Leave-one-out cross-validation**: Diese Methode ist für kleine Datenmengen gedacht
- Spezialfall der Kreuz-Validierung
- In jeder Runde wird nur eine Instanz beim Trainieren weggelassen, die dann zum Testen verwendet wird
- Wenn die Originaldaten *m* Instanzen haben, ist dies *m*-fache Kreuz-Validierung

Evaluierungsmethoden

- **Validierungsmenge**: die verfügbaren Daten werden in drei Teile geteilt,
  - Trainingsdaten,
  - Validierungsdaten und
  - Testdaten.
- Validierungsdaten werden oft zum Schätzen von Parametern des Lernverfahrens genutzt.
- Es werden die Parametereinstellungen genutzt, die auf den Validierungsdaten am besten abschneiden
- Kreuz-Validierung kann hier auch verwendet werden

Überblick

- Grundkonzepte
- Entscheidungsbäume
- Evaluierung von Klassifikatoren
- Lernen von Regeln
- Klassifikation mittels Assoziationsregeln
- Naïver Bayescher Klassifikator
- Naïve Bayes für Text Klassifikation
- Support Vektor Maschinen
- Ensemble-Methoden: Bagging und Boosting
- Zusammenfassung

Einführung

- Entscheidungsbaum kann in eine Regelmenge konvertiert werden
- Können Wenn-Dann Regeln auch direkt aus den Daten abgeleitet werden?
  - Ja.
  - Regel-Induktions-Systeme finden eine Sequenz von Regeln (auch Entscheidungsliste) für Klassifikation.
- Eine häufige Strategie ist sequentielles Überdecken.

Sequentielles Überdecken

- Lerne Regeln sequentiell nacheinander.
- Wenn eine Regel gelernt wurde, werden alle Trainingsbeispiele entfernt, die von der Regel überdeckt werden.
- Nur die verbleibenden Daten werden für die nachfolgenden Regeln genutzt.
- Der Prozess wird wiederholt bis ein Abbruchkriterium erfüllt ist.

Bemerkung: eine Regel überdeckt ein Beispiel, wenn das Beispiel die Voraussetzungen der Regel erfüllt.

- Zwei spezifische Algorithmen

Algorithmus 1: geordnete Regeln

Algorithm sequential-covering-1(D)

```
1   RuleList ← ∅;
2   Rule ← learn-one-rule-1(D);
3   while Rule is not NULL AND D ≠ ∅ do
4       RuleList ← insert Rule at the end of RuleList;
5       Remove from D the examples covered by Rule;
6       Rule ← learn-one-rule-1(D)
7   endwhile
8   insert a default class c at the end of RuleList, where c is the majority class
        in D;
9   return RuleList
```

- Erzeugter Klassifikator:

  <r_1, r_2, …, r_k, *default-class*>

Algorithmus 2: geordnete Klassen

```
Algorithm sequential-covering-2(D, C)
1   RuleList ← ∅;                          // empty rule set at the beginning
2   for each class c ∈C do
3       prepare data (Pos, Neg), where Pos contains all the examples of class c
            from D, and Neg contains the rest of the examples in D;
4       while Pos ≠ ∅ do
5           Rule ← learn-one-rule-2(Pos, Neg, c);
6           if Rule is NULL then
7             exit-while-loop
8           else  RuleList ← insert Rule at the end of RuleList;
9                 Remove examples covered by Rule from (Pos, Neg)
10          endif
11      endwhile
12  endfor
13  return RuleList
```

- Regeln derselben Klasse sind zusammen.

Algorithmus 1 & Algorithmus 2

- Unterschiede:
  - Algorithmus 2: Regeln derselben Klasse werden gemeinsam gefunden. Klassen sind sortiert. Man beginnt mit der kleinsten Klasse.
  - Algorithmus 1: In einer Iteration kann eine Regel einer beliebigen Klasse gefunden werden. Die Regeln sind in dieser Reihenfolge sortiert..
- Die Anwendung der Regeln ist bei beiden Algorithmen gleich
  - Für ein Testbeispiel werden die Regeln sequentiell getestet. Die erste Regel, die das Beispiel überdeckt, klassifiziert es.
  - Wenn keine Regel das Beispiel überdeckt, wird die Default Klasse genutzt, welche die Mehrheitsklasse ist.

Learn-one-rule-1 Funktion

- Nur kategorische Attribute werden diskutiert
- attributeValuePairs enthält alle möglichen Attribute-Wert Paare ($A_i = a_i$) der Daten.
- Iteration 1: Jedes Attribute-Wert Paar wird als Bedingung für eine Regel getestet, d.h. alle Regeln der Form $A_i = a_i \rightarrow c_j$. Die beste Regel wird ausgegeben,
  - Test: z.B., Entropie
  - Die k besten Regeln werden auch gespeichert für Beam Search (um mehr Suchraum zu testen) und werden new candidates genannt.

Learn-one-rule-1 Funktion

- In Iteration m, jede (m-1)-Bedingungsregel aus der new candidates Menge wird um ein Attribute-Wert Paar aus attributeValuePairs erweitert.
- Diese neuen Kandidaten Regeln wird wie die 1-Bedingungsregeln getestet.
  - Aktualisiere die beste Regel
  - Aktualisiere die k besten Regel
- Die Prozedur wird wiederholt bis das Abbruchkriterium erfüllt ist

Learn-one-rule-1 Algorithmus

```
   Function learn-one-rule-1(D),
1     BestCond ← ∅;                          // rule with no condition.
2     candidateCondSet ← {bestCond};
3     attributeValuePairs ← the set of all attribute-value pairs in D of the form
        (A_i op v), where A_i is an attribute and v is a value or an interval;
4     while candidateCondSet ≠ ∅ do
5         newCandidateCondSet ← ∅;
6         for each candidate cond in candidateCondSet do
7             for each attribute-value pair a in attributeValuePairs do
8                 newCond ← cond ∪ {a};
9                 newCandidateCondSet ← newCandidateCondSet ∪ {newCond}
10            endfor
11        endfor
12        remove duplicates and inconsistencies, e.g., {A_i = v_1, A_i = v_2};
13        for each candidate newCond in newCandidateCondSet do
14            if evaluation(newCond, D) > evaluation(BestCond, D) then
15                BestCond ← newCond;
16            endif
17        endfor
18        candidateCondSet ← the k best members of newCandidateCondSet
                according to the results of the evaluation function;
19    endwhile
20    if evaluation(BestCond, D) − evaluation(∅, D) > threshold then
21        return the rule: "BestCond → c" where is c the majority class of the
          data covered by BestCond;
22    else return NULL
23    endif
```

Learn-one-rule-2 Funktion

- Teile die Daten:
  - Pos -> GrowPos und PrunePos
  - Neg -> GrowNeg und PruneNeg
- Die Grow-Mengen werden zum Finden der Regeln genutzt (*BestRule*) und die Prune-Menge zu Streichen von Regeln.
- GrowRule arbeitet ähnlich wie learn-one-rule-1, aber die Klasse bleibt in diesem Fall fest. (Zur Erinnerung, der 2. Algorithmus findet erst alle Regeln einer Klasse (Pos) und geht dann zu nächsten Klasse

Learn-one-rule-2 Algorithmus

```
Function learn-one-rule-2(Pos, Neg, class)
1   split (Pos, Neg) into (GrowPos, GrowNeg) and (PrunePos, PruneNeg)
2   BestRule ← GrowRule(GrowPos, GrowNeg, class)          // grow a new rule
3   BestRule ← PruneRule(BestRule, PrunePos, PruneNeg)    // prune the rule
4   if the error rate of BestRule on (PrunePos, PruneNeg) exceeds 50% then
5      return NULL
6   endif
7   return BestRule
```

Regeltest learn-one-rule-2

- Eine teilweise entwickelte Regel sei:

  R: av_1, .., $av_k \rightarrow class$

  – wobei av_j eine Bedingung ist (ein Attribute-Wert Paar).
- Durch Hinzufügen einer neuen Bedingung av_{k+1}, entsteht Regel

  $R+$: av_1, .., av_k, $av_{k+1} \rightarrow class$.
- Die Testfunktion für $R+$ ist ein geändertes Information **gain** Kriterium

$$gain(R, R^+) = p_1 \times \left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right)$$

- Die Regel bis dem größten Zuwachs wird für eine Erweiterung behalten.

Regel-streichen in learn-one-rule-2

- Lösche testweise jede Teilmenge aus der Bedingung für *BestRule*, und lösche dann die Teilmenge, welche die Funktion maximiert:

$$v(BestRule, PrunePos, PruneNeg) = \frac{p - n}{p + n}$$

wobei *p* (*n*) ist die Anzahl der Beispiele in *PrunePos* (*PruneNeg*), die von der aktuellen Regel überdeckt werden (nach dem Löschen).

Diskussion

- Genauigkeit: wie bei Entscheidungsbäumen
- Effizienz: Laufen langsamer als Entscheidungsbäume weil
  - Um einer Regel zu erzeugen, werden viele Kandidaten getestet.
  - Wenn die Datenmenge und/oder die Anzahl der Attribut-Wert Paare groß sind, braucht der Alg. lange.
- Interpretation der Regeln: Kann problematisch sein, weil eine Regel gefunden wurde, nach die Daten, die durch vorhergehende Regeln überdeckt werden entfernt wurden. Deshalb sind die Regeln nicht unabhängig von einander.

Road Map

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Rule induction
- **Classification using association rules**
- Naïve Bayesian classification
- Naïve Bayes for text classification
- Support vector machines
- K-nearest neighbor
- Ensemble methods: Bagging and Boosting
- Summary

Three approaches

- Three main approaches of using association rules for classification.
  - Using class association rules to build classifiers
  - Using class association rules as attributes/features
  - Using normal association rules for classification

Using Class Association Rules

- **Classification:** mine a small set of rules existing in the data to form a classifier or predictor.
  - It has a target attribute: Class attribute
- **Association rules:** have no fixed target, but we can fix a target.
- Class association rules (CAR): has a target class attribute. E.g.,

  Own_house = true $\rightarrow$ Class =Yes [sup=6/15, conf=6/6]
  - CARs can obviously be used for classification.

Decision tree vs. CARs

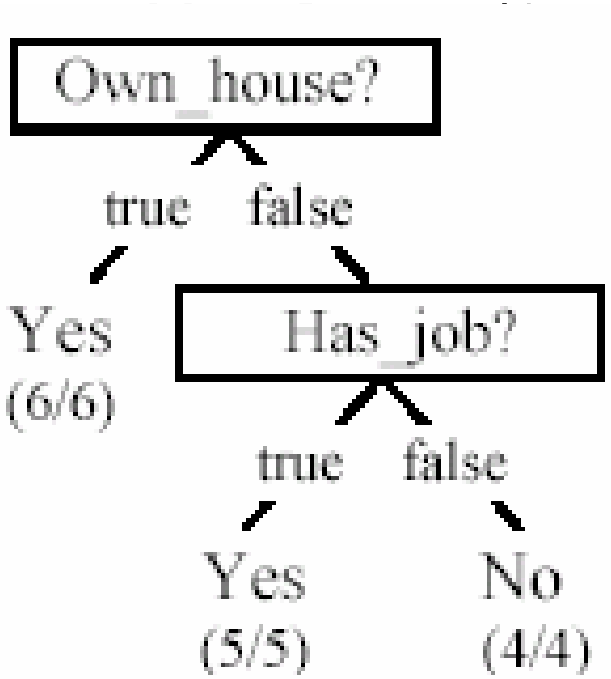
- **The decision tree below generates the following 3 rules**.

Own_house = true → Class =Yes [sup=6/15, conf=6/6]

Own_house = false, Has_job = true → conf=5/5]

Own_house = false, Has_job = false – conf=4/4]

- But there are many other rules that are not found by the decision tree



There are many more rules

Age = young, Has_job = true → Class=Yes [sup=2/15, conf=2/2]
Age = young, Has_job = false → Class=No [sup=3/15, conf=3/3]
Credit_Rating = fair → Class=No [sup=4/15, conf=4/4]
Credit_Rating = good → Class=Yes [sup=5/15, conf=5/6]

and many more, if we use minsup = 2/15 = 13.3% and minconf = 80%.

- CAR mining finds all of them.
- In many cases, rules not in the decision tree (or a rule list) may perform classification better.
- Such rules may also be actionable in practice

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	excellent	No
3	young	true	false	good	Yes
4	young	true	true	good	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

Decision tree vs. CARs (cont …)

- Association mining require discrete attributes. Decision tree learning uses both discrete and continuous attributes.
  - CAR mining requires continuous attributes discretized. There are several such algorithms.
- Decision tree is not constrained by minsup or minconf, and thus is able to find rules with very low support. Of course, such rules may be pruned due to the possible overfitting.

Considerations in CAR mining

- Multiple minimum class supports
  - Deal with imbalanced class distribution, e.g., some class is rare, 98% negative and 2% positive.
  - We can set the minsup(positive) = 0.2% and minsup(negative) = 2%.
  - If we are not interested in classification of negative class, we may not want to generate rules for negative class. We can set minsup(negative)=100% or more.
- Rule pruning may be performed.

Building classifiers

- There are many ways to build classifiers using CARs. Several existing systems available.
- Strongest rules: After CARs are mined, do nothing.
  - For each test case, we simply choose the most confident rule that covers the test case to classify it. Microsoft SQL Server has a similar method.
  - Or, using a combination of rules.
- Selecting a subset of Rules
  - used in the CBA system.

CBA: Rules are sorted first

Definition: Given two rules, r_i and r_j, $r_i \succ r_j$ (also called *ri* precedes r_j or r_i has a higher precedence than r_j) if

- the confidence of r_i is greater than that of r_j, or
- their confidences are the same, but the support of r_i is greater than that of r_j, or
- both the confidences and supports of r_i and r_j are the same, but r_i is generated earlier than r_j.

A CBA classifier *L* is of the form:

$L = <r_1, r_2, \ldots, r_k, default\text{-}class>$

Classifier building using CARs

```
   Algorithm CBA(S, D)
1  S = sort(S);                    // sorting is done according to the precedence ≻
2  RuleList = ∅;                   // the rule list classifier
3  for each rule r ∈ S in sequence do
4      if D ≠ ∅ AND r classifies at least one example in D correctly then
5          delete from D all training examples covered by r;
6          add r at the end of RuleList
7      end
8  end
9  add the majority class as the default class at the end of RuleList
```

- This algorithm is very inefficient
- CBA has a very efficient algorithm (quite sophisticated) that scans the data at most two times.

Using rules as features

- Most classification methods do not fully explore multi-attribute correlations, e.g., naïve Bayesian, decision trees, rules induction, etc.
- This method creates extra attributes to augment the original data by
  - Using the conditional parts of rules
  - Each rule forms an new attribute
  - If a data record satisfies the condition of a rule, the attribute value is 1, and 0 otherwise
- One can also use only rules as attributes
  - Throw away the original data

Using normal association rules for classification

- A widely used approach
- Main approach: strongest rules
- Main application
  - Recommendation systems in e-commerce Web site (e.g., amazon.com).
  - Each rule consequent is the recommended item.
- Major advantage: any item can be predicted.
- Main issue:
  - Coverage: rare item rules are not found using classic algo.
  - Multiple min supports and support difference constraint help a great deal.