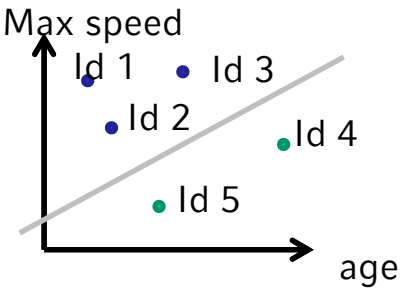
Chapter 6: Classification

1) Introduction
   - Classification problem, evaluation of classifiers, prediction
2) Bayesian Classifiers
   - Bayes classifier, naive Bayes classifier, applications
3) Linear discriminant functions & SVM

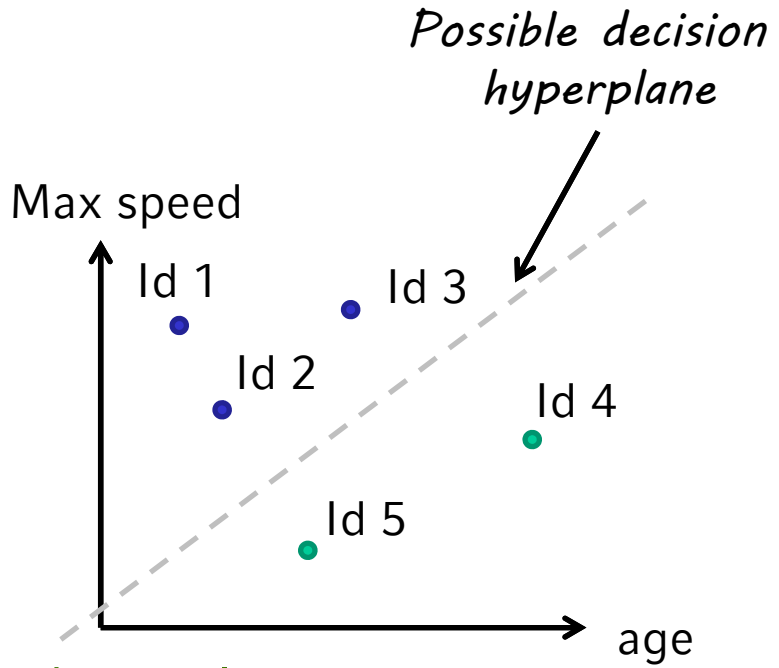


4) Decision Tree Classifiers
   - Basic notions, split strategies, overfitting, pruning of decision trees
5) Nearest Neighbor Classifier
   - Basic notions, choice of parameters, applications
6) Ensemble Classification

Linear discriminant function classifier

- Example

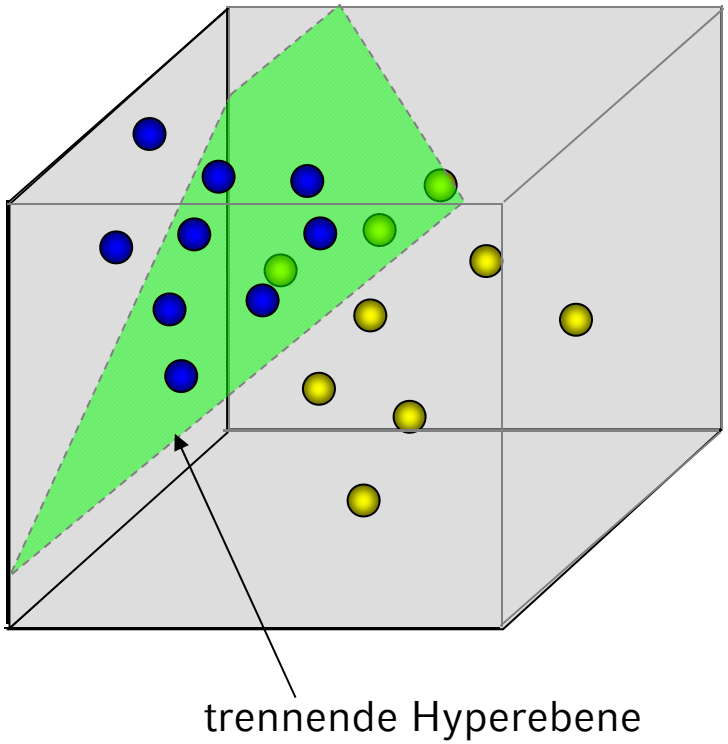
ID	age	car type	Max speed	risk
1	23	family	180	high
2	17	sportive	240	high
3	43	sportive	246	high
4	68	family	173	low
5	32	truck	110	low



- Idea: separate points of two classes by a hyperplane
  - I.e., classification model is a hyperplane
  - Points of one class in one half space, points of second class are in the other half space
- Questions:
  - How to formalize the classifier?
  - How to find optimal parameters of the model?

Support Vector Machines

Eine der bekanntesten Klassifikatoren, die lineare Separation umsetzen



- Vektoren in R^d repräsentieren Objekte.
- Objekte gehören zu genau einer von je 2 Klassen

Klassifikation durch lineare Separation:

- Suche Hyperebene, die beide Klassen „maximal stabil“ voneinander trennt.

- Ordne unbekannte Elemente der Seite der Ebene zu, auf der sie sich befinden.

Support Vector Machines

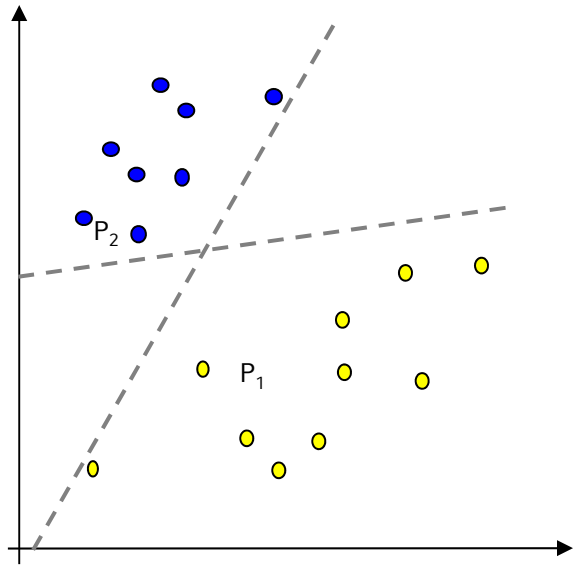
Probleme bei linearer Separation:

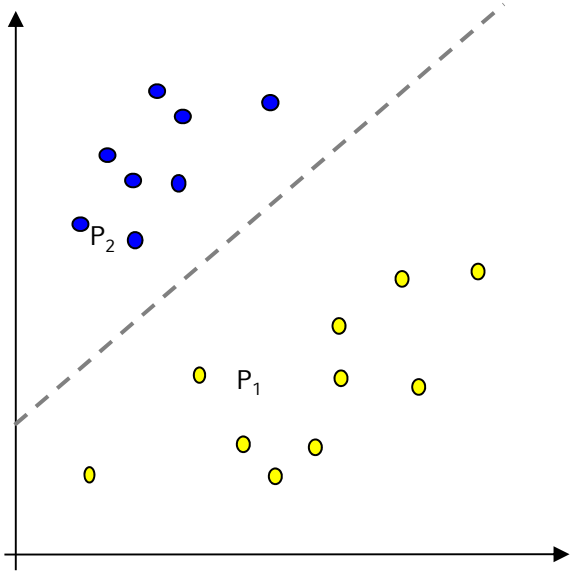
- Was ist die „maximal stabile“ Hyperebene und wie berechnet man sie effizient?
- Klassen nicht immer linear trennbar.
- Berechnung von Hyperebenen nach Auswahl sehr aufwendig.
- Einschränkung auf 2 Klassen.
- ...

⇨ Lösungen dieser Probleme mit Support Vector Machines(SVMs) [Vapnik 1979 u. 1995].

Maximum Margin Hyperplane

Problem: Hyperebene, die P_1 und P_2 trennt, ist nicht eindeutig.
$\Rightarrow$ Welche Hyperebene ist für die Separation die Beste ?

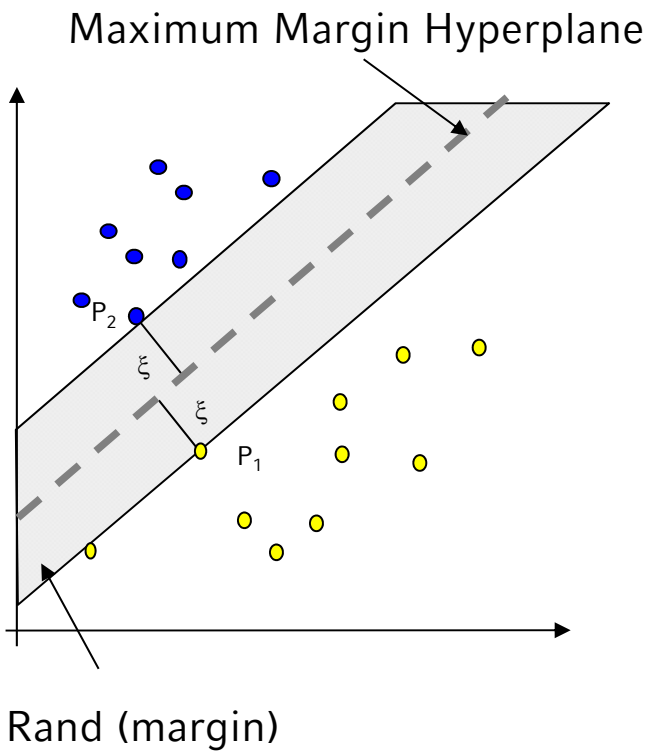




Kriterien:
- Stabilität beim Einfügen
- Abstand zu den Objekten beider Klassen

Maximum Margin Hyperplane

- Lineare Separation mit der *„Maximum Margin Hyperplane“*



- Abstand zu Punkten aus beiden Mengen ist maximal, d.h. mind. ξ
- Wahrscheinlichkeit, dass beim Einfügen die trennende Hyperebene verschoben werden muss, ist minimal
- generalisiert am besten

$\Rightarrow$ Maximum Margin Hyperplane (MMH) ist „maximal stabil“

MMH ist nur von Punkten P_i abhängig, die Abstand ξ zur Ebene aufweisen.

$\Rightarrow$ P_i heißt Support Vector

Maximum Margin Hyperplane

- Zusammenfassung der Schreibweisen der benötigten algebraischen Konstrukte für Featurespace *FS*:

- Skalarprodukt zweier Vektoren: $\langle \vec{x}, \vec{y} \rangle, \vec{x}, \vec{y} \in FS$
- z.B. $\langle \vec{x}, \vec{y} \rangle = \sum_{i=1}^{d} (x_i \cdot y_i)$ kanonisches Skalarprodukt

- Beschreibung einer Hyperebene: $H(\vec{w}, b) = \left\{ \vec{x} \in FS \,\middle|\, 0 = \langle \vec{w}, \vec{x} \rangle + b \right\}$

- Abstand eines Vektors zur Ebene: $dist(\vec{x}, H(\vec{w}, b)) = \left| \frac{1}{\sqrt{\langle \vec{w}, \vec{w} \rangle}} \langle \vec{w}, \vec{x} \rangle + b \right|$

Maximum Margin Hyperplane

Berechnung der Maximum Margin Hyperplane

1. Bedingung: kein Klassifikationsfehler (Klasse1: $y_i=1$,Klasse 2:$y_i=-1$)

$$\left.\begin{aligned}(y_i=-1)\Rightarrow[\langle\vec{w},\vec{x_i}\rangle+b]<0\\(y_i=1)\Rightarrow[\langle\vec{w},\vec{x_i}\rangle+b]>0\end{aligned}\right\}\Leftrightarrow y_i\left[\langle\vec{w},\vec{x_i}\rangle+b\right]>0$$

2. Bedingung: Maximaler Rand (Margin)

maximiere: $\xi=\min_{\vec{x}_i\in TR}\left|\frac{1}{\sqrt{\langle\vec{w},\vec{w}\rangle}}\left(\langle\vec{w},\vec{x_i}\rangle+b\right)\right|$ (Abstand von x_i zur Ebene $H(\vec{w},b)$)

oder

maximiere: ξ , so dass $\left[y_i\left(\frac{1}{\sqrt{\langle\vec{w},\vec{w}\rangle}}\left[\langle\vec{w},\vec{x_i}\rangle+b\right]\right)\geq\xi\right]$ für $\forall i\in[1..n]$

Maximum Margin Hyperplane

- maximiere ξ in $\left[y_i \left(\frac{1}{\sqrt{\langle \vec{w}, \vec{w} \rangle}} \left[\langle \vec{w}, \vec{x}_i \rangle + b \right] \right) \geq \xi \right]$; für $\forall i \in [1..n]$

- Setze $\frac{1}{\sqrt{\langle \vec{w}, \vec{w} \rangle}} = \xi$: max. $\frac{1}{\sqrt{\langle \vec{w}, \vec{w} \rangle}}$, mit $\left(y_i \cdot \left(\xi \cdot \left(\langle \vec{w}, \vec{x}_i \rangle + b \right) \right) \geq \xi \right)$ $\forall i \in [1..n]$

- $\Rightarrow$ max. $\frac{1}{\sqrt{\langle \vec{w}, \vec{w} \rangle}}$, mit $\left(y_i \left(\langle \vec{w}, \vec{x}_i \rangle + b \right) \geq 1 \right)$ $\forall i \in [1..n]$

- Statt $\frac{1}{\sqrt{\langle \vec{w}, \vec{w} \rangle}}$ invertiere, quadriere und minimiere das Ergebnis:

Primäres OP: minimiere $J(\vec{w}, b) = \langle \vec{w}, \vec{w} \rangle$

unter Nebenbedingung für $\forall i \in [1..n]$ sei $\left(y_i \left(\langle \vec{w}, \vec{x}_i \rangle + b \right) \geq 1 \right)$

Maximum Margin Hyperplane

Zur Berechnung wird das primäre Optimierungsproblem in ein duales OP überführt.
(Umformulierung in Form mit Langrange Multiplikatoren nach Karush-Kuhn-Tucker)

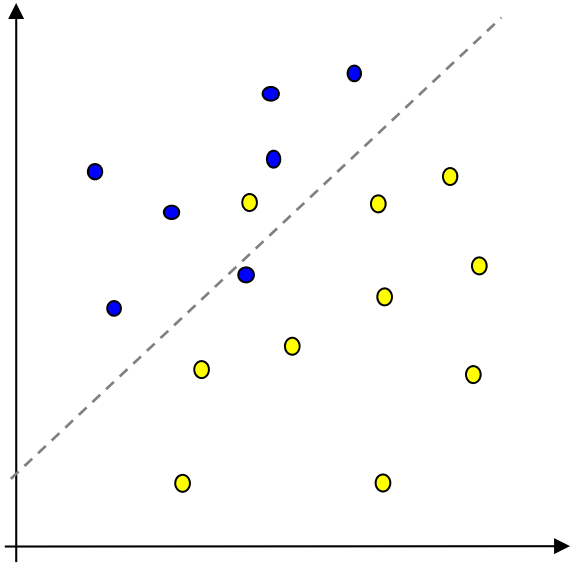
Duales OP: maximiere $$L(\vec{\alpha}) = \left(\sum_{i=1}^{n} \alpha_i \right) - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \left\langle \vec{x}_i, \vec{x}_j \right\rangle$$

unter Bedingung $$\sum_{i=1}^{n} \alpha_i \cdot y_i = 0 \quad , 0 \leq \alpha_i \text{ und } \quad \vec{\alpha} \in \Re^n$$

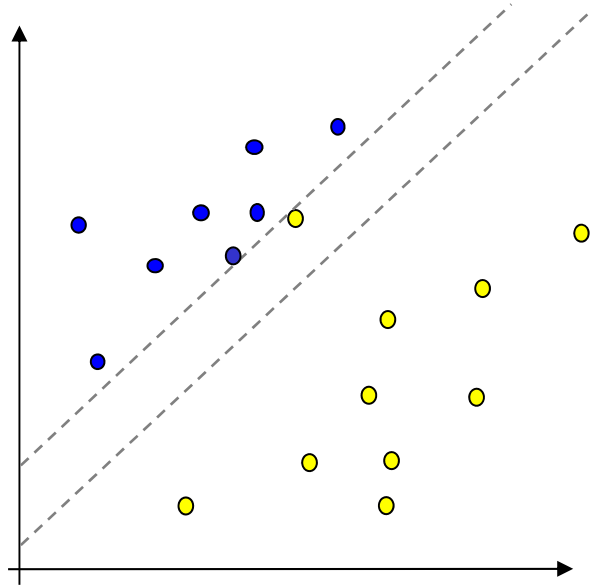
⇒ Lösung des Problems mit Algorithmen aus der Optimierungstheorie
⇒ bis jetzt nur linear separierbarer Fall: Soft Margin Optimierung
⇒ Einführung von Kernelfunktionen zur Steigerung der Kapazität

Soft Margin

Behandlung nicht linear trennbarer Daten:
Soft Margin Optimierung



Daten nicht separierbar

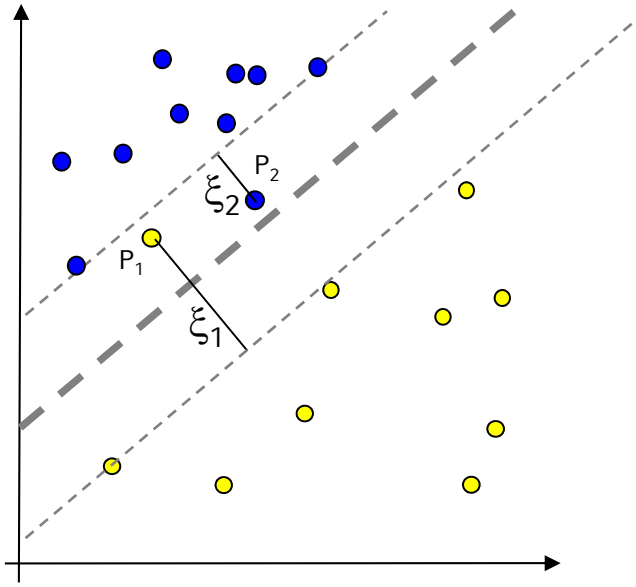


vollständige Separation ist nicht optimal

$\Rightarrow$ Trade-Off zwischen Trainingsfehler und Breite des Randes

Soft Margin

Betrachte beim Optimieren zusätzlich noch die Anzahl der Trainingsfehler.



- ξ_i ist der Abstand von P_i zum Rand (wird auch Slack-Variable genannt)
- C reguliert den Einfluss eines einzelnen Trainingsvektors

Primäres OP :

minimiere $J(\vec{w}, b, \vec{\xi}) = \frac{1}{2}\left\langle \vec{w}, \vec{w} \right\rangle + C \cdot \sum_{i=1}^{n} \xi_i$

unter Nebenbedingung für $\forall i \in [1..n]$ sei $y_i\left(\left\langle \vec{w}, \vec{x_i} \right\rangle + b\right) \geq 1 - \xi_i$ und $\xi_i \geq 0$

$\Rightarrow$ Primäres Optimierungsproblem unter weichen Grenzen (Soft Margin)

Soft Margin

Das duale OP mit Lagrange Multiplikatoren verändert sich wie folgt:

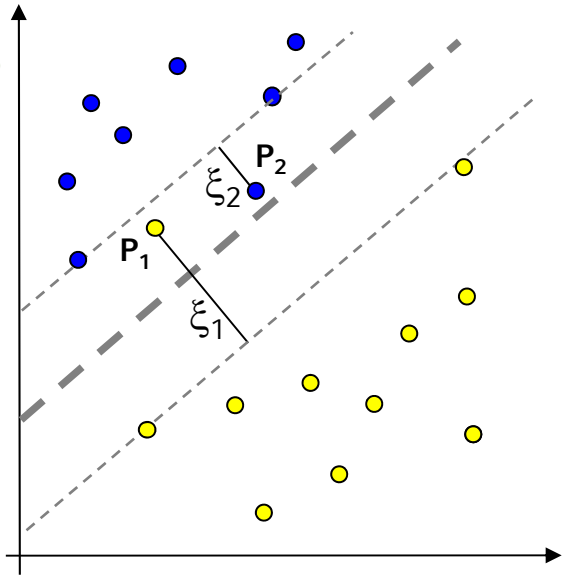
Duales OP: maximiere $L(\vec{\alpha}) = \left(\sum_{i=1}^{n} \alpha_i \right) - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \left\langle \vec{x}_i, \vec{x}_j \right\rangle$

mit Bedingung $\sum_{i=1}^{n} \alpha_i \cdot y_i = 0$ und $0 \leq \alpha_i \leq \mathbf{C}$

- $0 < \alpha_i < C \Leftrightarrow$ Support Vektor mit $\xi_i = 0$
- $\alpha_i = C \Leftrightarrow$ Support Vektor mit $\xi_i > 0$
- $\alpha_i = 0$ sonst

Entscheidungsregel:

$$h(\vec{x}) = sign\left(\sum_{x_i \in SV} \alpha_i \cdot y_i \left\langle \vec{x}_i, \vec{x} \right\rangle + b \right)$$

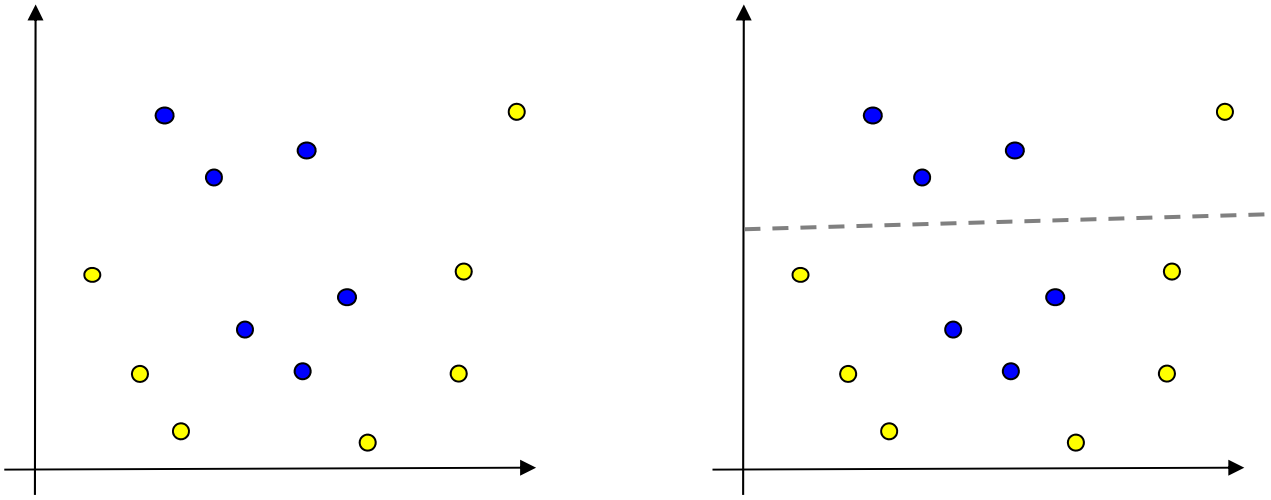


Kernel Machines

Lernen bei nicht linear trennbaren Datenmengen

Problem: Bei realen Problemen ist häufig keine lineare Separation mit hoher Klassifikationsgenauigkeit mehr möglich.

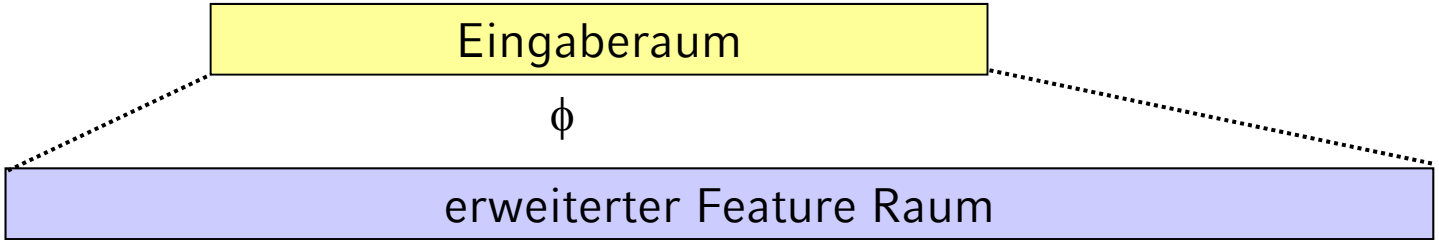
Idee: Transformiere Daten in einen nicht linearen Feature-Raum und versuche sie im neuen Raum linear zu separieren.
(Erweiterung des Hypothesenraumes)



Beispiel: quadratische Transformation

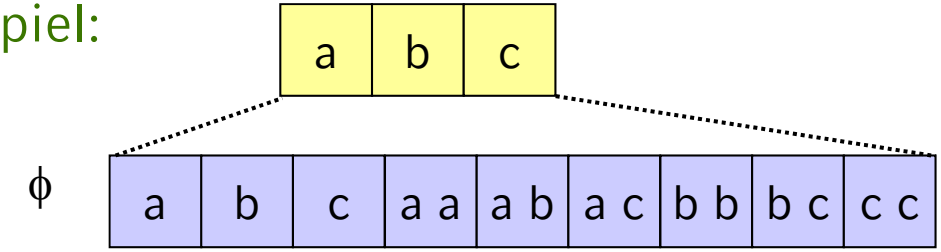
Kernel Machines

Erweiterung der Hypothesenraumes



⇒ Versuche jetzt in erweitertem Feature Raum linear zu separieren

Beispiel:

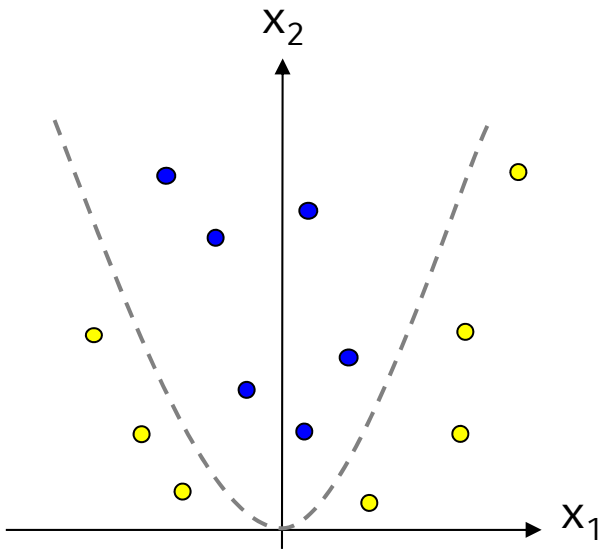


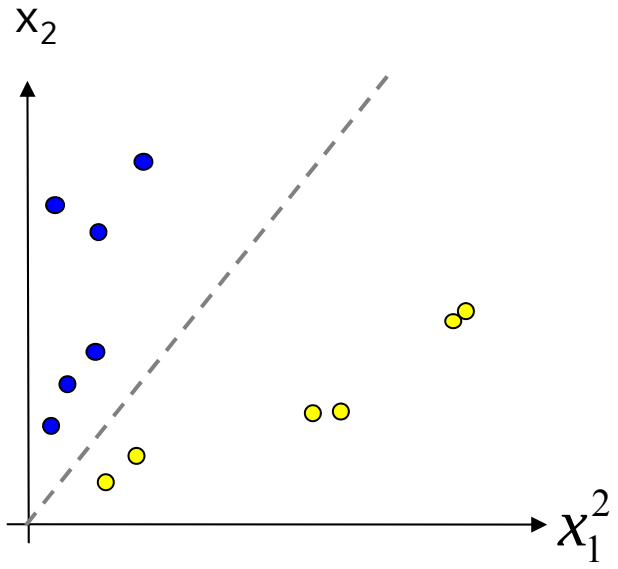
hier: Eine Hyperebene im erweiterten Feature Raum ist ein Polynom 2. Grades im Eingaberaum.

Kernel Machines

Eingaberaum: $\vec{x} = (x_1, x_2)$ (2 Attribute)

im erweiterten Raum: $\phi(\vec{x}) = (x_1^2, x_2^2, \sqrt{2} \cdot x_1, \sqrt{2} \cdot x_2, \sqrt{2} \cdot x_1 \cdot x_2, 1)$ (6 Attribute)





Kernel Machines

- Einführung eines Kernels ⇔ (implizite) Featuretransformation mittels

$$\phi(\vec{x}): FS_{alt} \longrightarrow FS_{neu}$$

Duales OP: maximiere

$$L(\vec{\alpha}) = \left(\sum_{i=1}^{n} \alpha_i\right) - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \left\langle \phi(\vec{x}_i), \phi(\vec{x}_j) \right\rangle$$

mit Bedingung $\sum_{i=1}^{n} \alpha_i \cdot y_i = 0$ und $0 \leq \alpha_i \leq C$

Zusätzliche Featuretransformation wirkt sich nur auf das Skalarprodukt der Trainingsvektoren aus. ⇒ Kernel K ist eine Funktion mit:

$$K_\phi\left(\vec{x}_i, \vec{x}_j\right) = \left\langle \phi(\vec{x}_i), \phi(\vec{x}_j) \right\rangle$$

Wann ist eine Funktion K(x,y) ein Kernel ?

Wenn die **Kernel-Matrix** (Gram Matrix) KM

$$KM(K) = \begin{pmatrix} K(\vec{x}_1, \vec{x}_1) & .. & K(\vec{x}_1, \vec{x}_n) \\ .. & .. & .. \\ K(\vec{x}_n, \vec{x}_1) & .. & K(\vec{x}_n, \vec{x}_n) \end{pmatrix}$$

positiv (semi-) definit ist, also keine negativen Eigenwerte besitzt, dann ist K(x,y) ein Kernel (Mercer's Theorem)

Notwendige Bedingungen:

- $K_\phi\left(\vec{x}, \vec{y}\right) = \left\langle \phi(\vec{x}), \phi(\vec{y}) \right\rangle = \left\langle \phi(\vec{y}), \phi(\vec{x}) \right\rangle = K_\phi\left(\vec{y}, \vec{x}\right)$ (Symmetrie)
- $K_\phi\left(\vec{x}, \vec{y}\right)^2 \leq K_\phi\left(\vec{x}, \vec{x}\right) \cdot K_\phi\left(\vec{y}, \vec{y}\right)$ (Cauchy-Schwarz)

Symmetrie und Cauchy-Schwarz sind keine hinreichenden Bedingungen!

Kernel Machines

einige Regeln zur Kombination vom Kerneln:

$$K\left(\vec{x},\vec{y}\right)=K_1\left(\vec{x},\vec{y}\right)\cdot K_2\left(\vec{x},\vec{y}\right)$$

$$K\left(\vec{x},\vec{y}\right)=K_1\left(\vec{x},\vec{y}\right)+K_2\left(\vec{x},\vec{y}\right)$$

$$K\left(\vec{x},\vec{y}\right)=a\cdot K_1\left(\vec{x},\vec{y}\right)$$

$$K\left(\vec{x},\vec{y}\right)=x^T\cdot B\cdot y$$

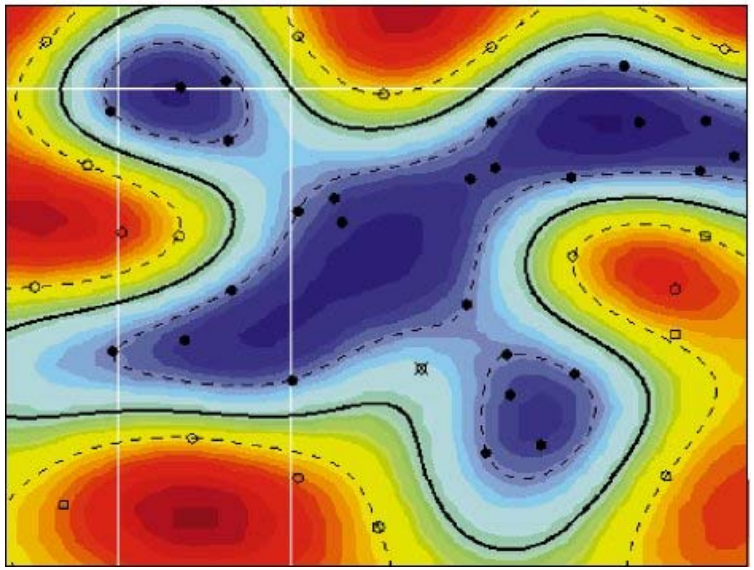
für $\mathbf{K_1}$, $\mathbf{K_2}$ Kernelfunktionen, **a** eine positive Konstante und **B** eine symmetrische positiv semi-definite Matrix.

Kernel Machines

Beispiele für verwendete Kernel-Funktionen:

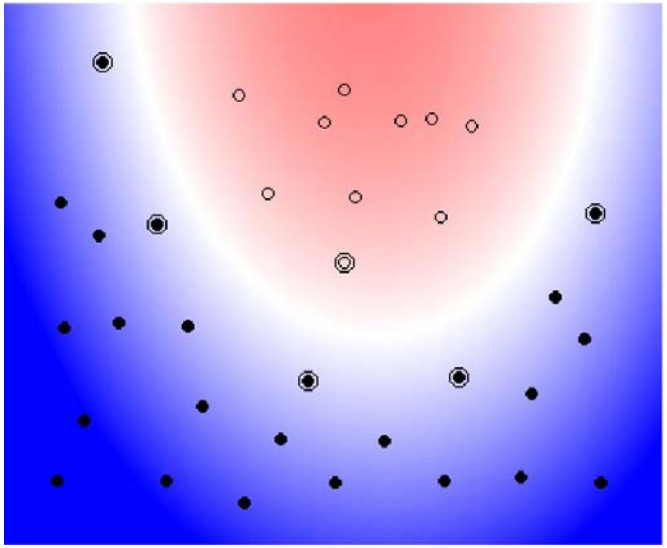
- linear: $K(\vec{x}, \vec{y}) = \langle \vec{x}, \vec{y} \rangle$
- polynomiell: $K(\vec{x}, \vec{y}) = \left(\langle \vec{x}, \vec{y} \rangle + c \right)^d$
- Radiale Basisfunktionen: $K(\vec{x}, \vec{y}) = \exp\left(-\gamma \cdot \left| \vec{x} - \vec{y} \right|^2 \right)$
- Gauss Kernel: $K(\vec{x}, \vec{y}) = \exp\left(-\frac{\left\| \vec{x} - \vec{y} \right\|^2}{2\sigma^2} \right)$
- Sigmoid: $K(\vec{x}, \vec{y}) = \tanh\left(\gamma \cdot \langle \vec{x}, \vec{y} \rangle + c \right)$

Kernel Machines



Radial Basis Kernel

Polynomieller Kernel (Grad 2)



Training einer SVM

zu lösen ist folgendes Problem:

Duales OP: maximiere

$$L(\vec{\alpha}) = \left(\sum_{i=1}^{n} \alpha_i\right) - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot K\left(\vec{x}_i, \vec{x}_j\right)$$

mit Bedingung $\sum_{i=1}^{n} \alpha_i \cdot y_i = 0$ und $0 \leq \alpha_i \leq C$

oder

$$\max \begin{bmatrix} 1 \\ .. \\ 1 \end{bmatrix}^T \cdot \begin{bmatrix} \alpha_1 \\ .. \\ \alpha_n \end{bmatrix} - \frac{1}{2} \begin{bmatrix} \alpha_1 \\ .. \\ \alpha_n \end{bmatrix}^T \cdot \begin{bmatrix} y_1 y_1 K(\vec{x}_1, \vec{x}_1) & .. & y_1 y_n K(\vec{x}_1, \vec{x}_n) \\ .. & .. & .. \\ y_n y_1 K(\vec{x}_n, \vec{x}_1) & .. & y_n y_n K(\vec{x}_n, \vec{x}_n) \end{bmatrix} \cdot \begin{bmatrix} \alpha_1 \\ .. \\ \alpha_n \end{bmatrix}$$

mit Bedingung $\sum_{i=1}^{n} \alpha_i \cdot y_i = 0$ und $0 \leq \alpha_i \leq C$

Training einer SVM

zur Lösung:

- Standardalgorithmen aus der Optimierungstheorie für konvexe quadratische Probleme
- für große Trainingsmengen numerische Algorithmen notwendig
- es existieren einige Spezialalgorithmen für SVM-Training:
  - Chunking / Decomposition
  - Sequential Minimal Optimisation (SMO)

Multi-Class SVMs

- Bisher SVMs nur anwendbar auf 2 Klassen Probleme !!

- **Idee**: Kombination mehrere 2-Klassen SVMs zu Klassifikatoren, die beliebig viele Klassen unterscheiden können.

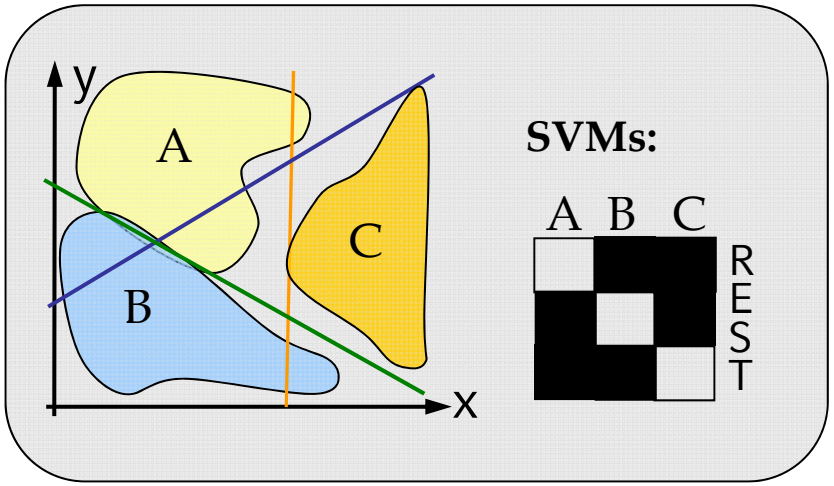
Multi-Class SVMs

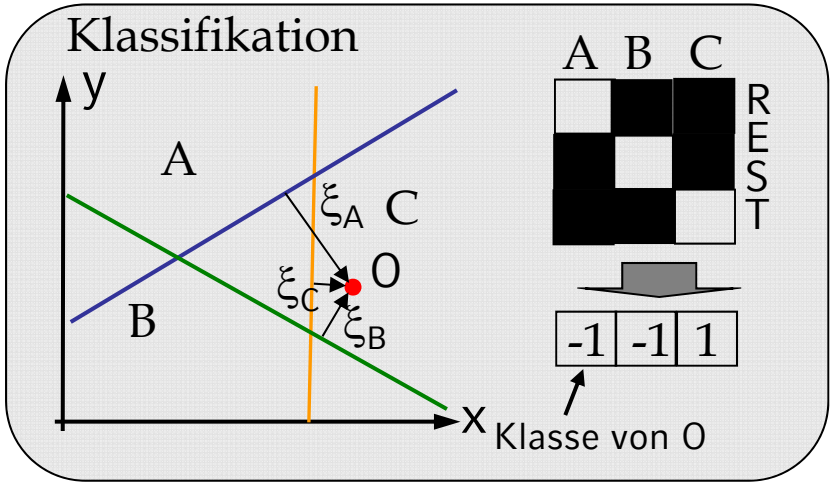
2 klassische Ansätze:

⇒ Unterscheide jede Klasse von allen anderen
(1-versus-Rest)

⇒ Unterscheide je 2 Klassen
(1-versus-1)

1-versus-Rest Ansatz





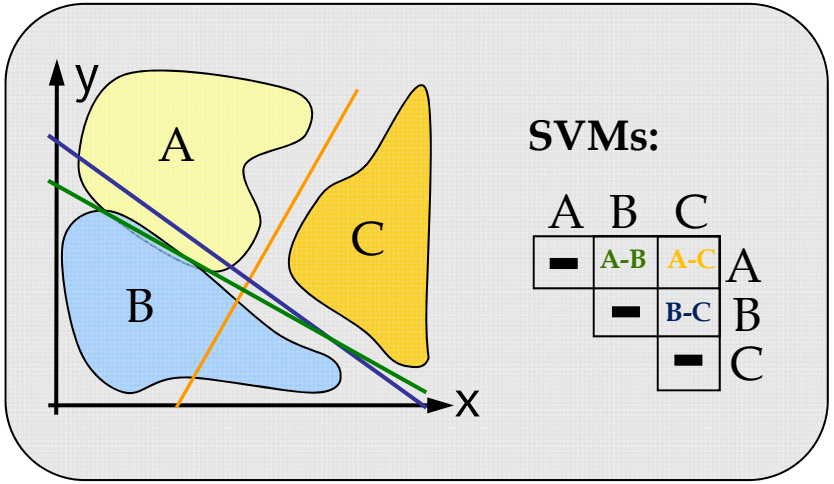
- 1-versus-Rest Ansatz : 1 SVM für jede Klasse.
- SVM trennt jedes Klasse von Vereinigung aller anderen Klassen ab
- Klassifiziere O mit allen Basis-SVMs.

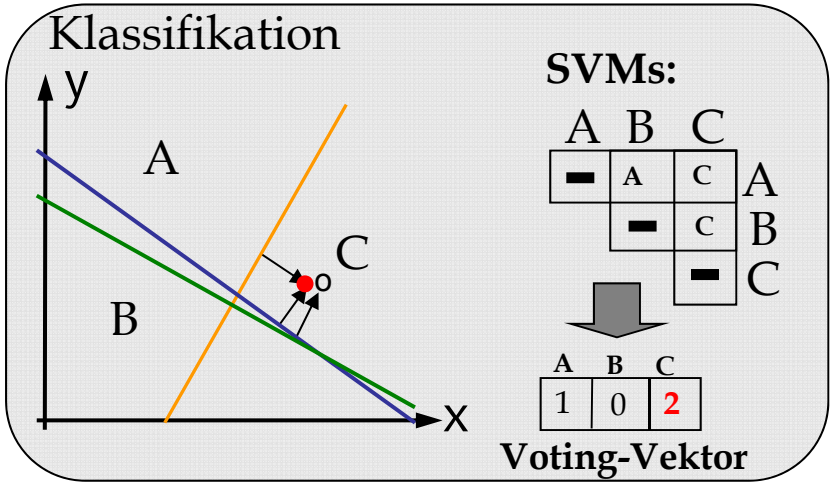
  $\Rightarrow$ **Multiple Klassenzugehörigkeit möglich (Multi-Classification)**

  oder

  Entscheidung für die Klasse, bei der Abstand ξ_i am größten ist.

1-versus-1 Ansatz





- 1-versus-1 Ansatz : 1 SVM für jedes Paar von Klassen.
- Klassifikation von Objekt O:
  1. Klassifiziere O mit allen Basis-SVMs.
  2. Zähle “Stimmen”(Votes) für jede Klasse.
- Maximale Anzahl an Votes => Ergebnis.

Vergleich 1-versus-rest und 1-versus-1

Kriterium	1-versus-rest	1-versus-1
Aufwand Training	linear zur Anzahl der Klassen ($O(\|K\|)$)	Quadratisch zur anzahl der Klassen ($O(\|K\|^2)$)
Aufwand Klassifikation	linear zur Anzahl der Klassen ($O(\|K\|)$)	Quadratisch zur Anzahl der Klassen ($O(\|K\|^2)$) **Verbesserung:** „Decision Directed Acyclic Graphs“ Klassifikation in $O(\|K\|)$ [Platt,Christianini 1999]
Genauigkeit	tendenziell schlechter	tendenziell höher, (nutzt wissen über unterschiedliche Klassen besser aus)

Support Vector Machines

- *Diskussion*

- \+ erzeugt Klassifikatoren mit hoher Genauigkeit
- \+ verhältnismäßig schwache Tendenz zu Overfitting
- (Begründung durch Generalisierungtheorie)
- \+ effiziente Klassifikation neuer Objekte
- \+ kompakte Modelle

- unter Umständen lange Trainingszeiten
- aufwendige Implementierung
- gefundene Modelle schwer zu deuten

Support Vector Machines

Literatur:

- C. Cortes, V. Vapnik: Support-vector networks.
  Machine Learning, 20:273-297, November 1995.

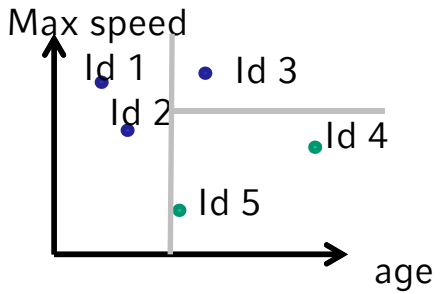
- C.J.C. Burges: A tutorial on support vector machines for pattern recognition.
  Data Mining and Knowledge Discovery, 2(2):121-167,1998.

- T. Joachims: Text categorisation with Support Vector Machines.
  in Proceedings of European Conference on Machine Learning (ECML), 1998.

- N. Cristianini, J. Shawe-Taylor: An Introduction to Support Vector Machines and other kernel-based learning methods.
  Cambridge University Press 2000.

Chapter 6: Classification

1) Introduction
   - Classification problem, evaluation of classifiers, prediction
2) Bayesian Classifiers
   - Bayes classifier, naive Bayes classifier, applications
3) Linear discriminant functions & SVM
4) Decision Tree Classifiers
   - Basic notions, split strategies, overfitting, pruning of decision trees



5) Nearest Neighbor Classifier
   - Basic notions, choice of parameters, applications
6) Ensemble Classification

Decision Tree Classifiers

- Approximating discrete-valued target function
- Learned function is represented as a tree:
  - A flow-chart-like tree structure
  - Internal node denotes a test on an attribute
  - Branch represents an outcome of the test
  - Leaf nodes represent class labels or class distribution
- Learned tree can be transformed into IF-THEN rules

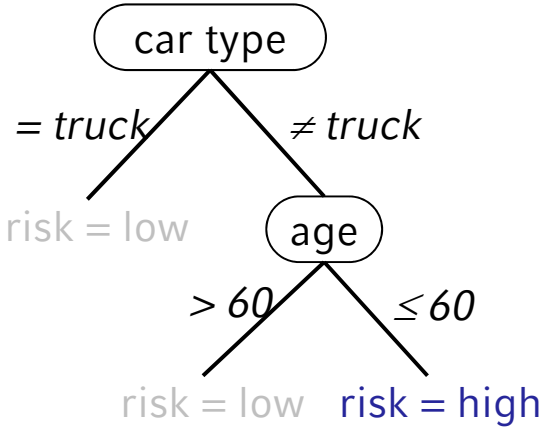
IF car_type = truck THEN risk = low
IF car_type ≠ truck AND age > 60 THEN risk = low
IF car_type ≠ truck AND age ≤ 60 THEN risk = high

- Advantages:
  - Decision trees represent explicit knowledge
  - Decision trees are intuitive to most users

training data

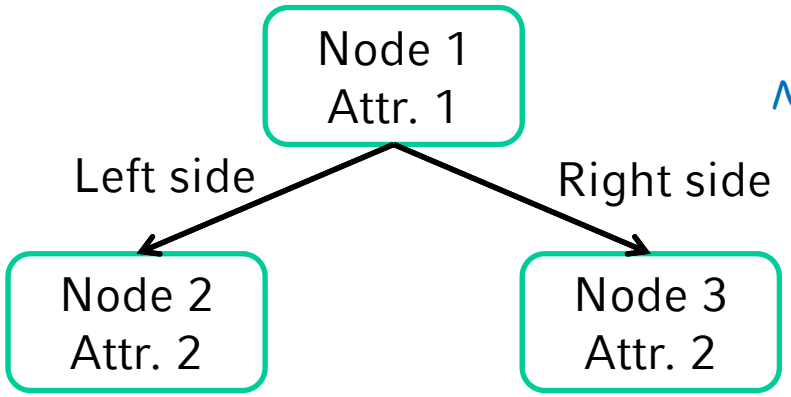
ID	age	car type	risk
1	23	family	high
2	17	sportive	high
3	43	sportive	high
4	68	family	low
5	32	truck	low

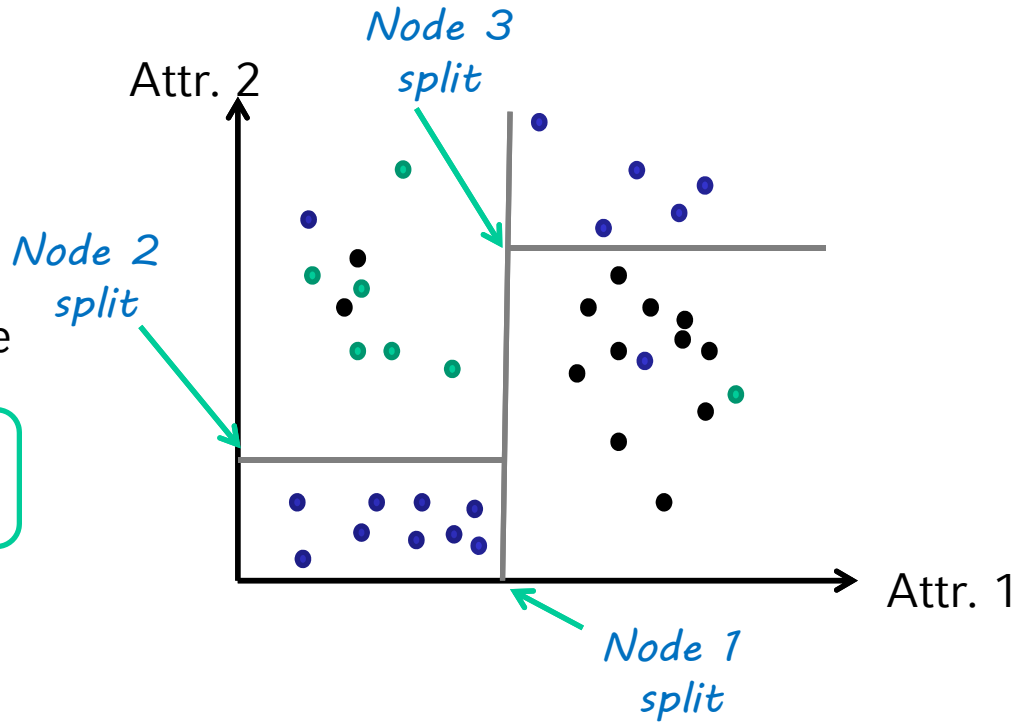
learned decision tree



Decision Tree Classifier: Splits

- Each tree node defines an axis-parallel (d-1)-dimensional hyper plane, that splits the domain space
- Goal: find such splits which lead to as homogenous groups as possible





Decision Tree Classifiers: Basics

- Decision tree generation (training phase) consists of two phases
  1) Tree construction
     - At start, all the training examples are at the root
     - Partition examples recursively based on selected attributes
  2) Tree pruning
     - Identify and remove branches that reflect noise or outliers

- Use of decision tree: Classifying an unknown sample
  - Traverse the tree and test the attribute values of the sample against the decision tree
  - Assign the class label of the respective leaf to the query object

Algorithm for Decision Tree Construction

- Basic algorithm (a greedy algorithm)
  - Tree is created in a top-down recursive divide-and-conquer manner
  - Attributes may be categorical or continuous-valued
  - At start, all the training examples are assigned to the root node
  - Recursively partition the examples at each node and push them down to the new nodes
    - Select test attributes and determine split points or split sets for the respective values on the basis of a heuristic or statistical measure (*split strategy,* e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Algorithm for Decision Tree Construction

- Most algorithms are versions of this basic algorithm (greedy, top-down)
  - E.g.: ID3[Q86], or its successor C4.5[Q96]

ID3(*Examples*, *TargetAttr*, *Attributes*) //specialized to learn boolean-valued functions

Create a *Root* node for the tree;

If all *Examples* are positive, return *Root* with label = + ;

If all *Examples* are negative, return *Root* with label = - ;

If *Attributes*=$\emptyset$, return *Root* with label = most common value of *TargetAttr* in *Examples*;

Else

A=the 'best' decision attribute for next node

how to determine the 'best' attribute ?

Assign A as decision attribute for *Root*

For each possible value v_i of A:

how to split the possible values ?

Generate branch corresponding to test $A = v_i$;

$Examples_{v_i}$= examples that have value v_i for A;

If $Examples_{v_i} = \emptyset$, add leaf node with label = most common value of *TargetAttr* in *Examples*;

Else add subtree ID3($Examples_{v_i}$, *TargetAttr*, *Attributes*\{A});

[Q86] J.R. Quinlan. *Induction of decision trees.* Machine Learnin, 1(1), pages 81-106, 1986.
[Q96] J. R. Quinlan. *Bagging, boosting, and c4.5.* In Proc. 13th Natl. Conf. on Artificial Intelligence (AAAI'96), pages 725-730, 1996.

Example: Decision for „playing_tennis“

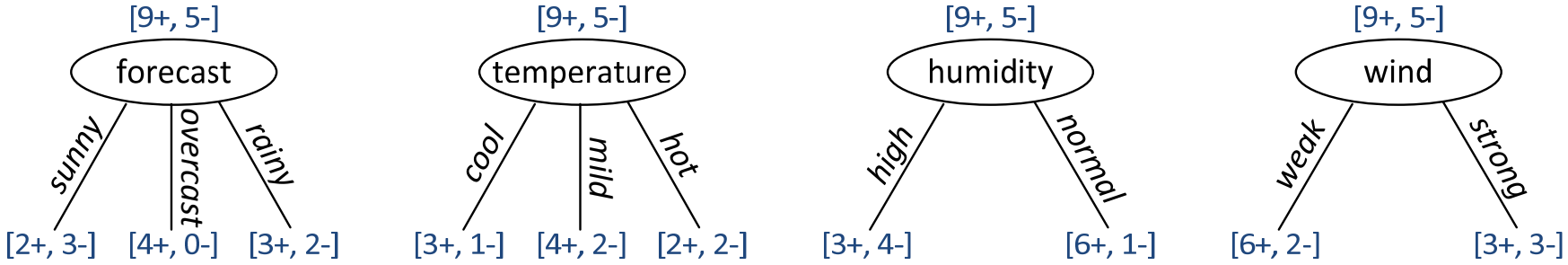
- Query: How about playing tennis today?
- Training data:

day	forecast	temperature	humidity	wind	tennis decision
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rainy	mild	high	weak	yes
5	rainy	cool	normal	weak	yes
6	rainy	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rainy	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rainy	mild	high	strong	no

- Build decision tree …

Split Strategies: Quality of Splits

- Given
  - a set T of training objects
  - a (disjoint, complete) partitioning $T_1, T_2, \ldots, T_m$ of T
  - the relative frequencies p_i of class c_i in T *and in the partition* $T_1, \ldots, T_m$

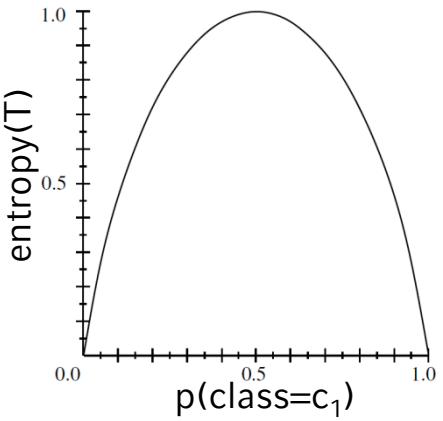


- Wanted
  - a measure for the heterogeneity of a set S of training objects with respect to the class membership
  - a split of T into partitions $T_1, T_2, \ldots, T_m$ such that the heterogeneity is minimized
- Proposals: Information gain, Gini index, Misclassification error

Attribute Selection Measures: Information Gain

- used in ID3 / C4.5
- Entropy
  - minimum number of bits to encode a message that contains the class label of a random training object
  - the *entropy* of a set T of training objects is defined as follows:

for two classes:



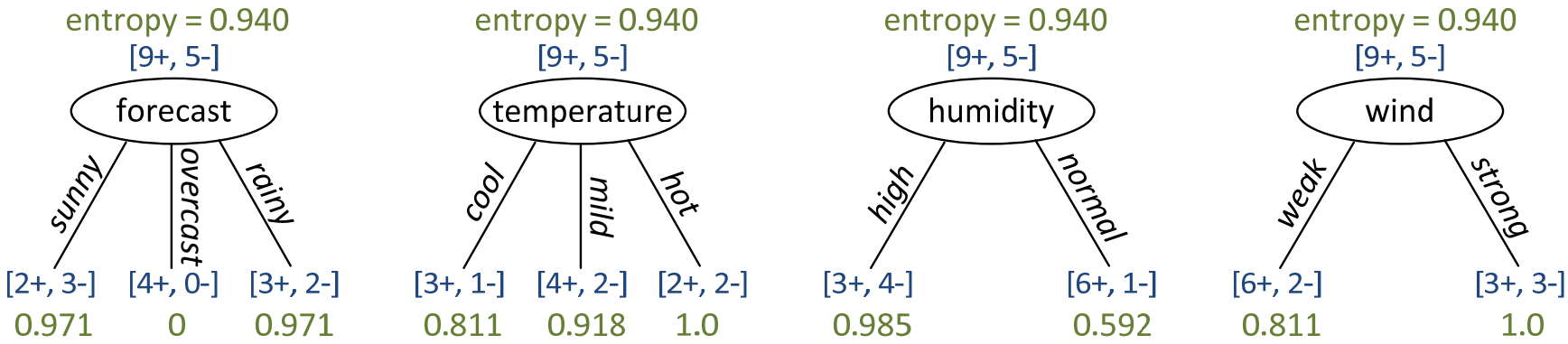
$$entropy(T) = -\sum_{i=1}^{k} p_i \cdot \log_2 p_i$$

for k classes c_i with frequencies p_i

  - entropy(T) = 0 if $p_i = 1$ for any class c_i
  - entropy (T) = 1 if there are $k = 2$ classes with $p_i = ½$ for each i
- Let A be the attribute that induced the partitioning $T_1, T_2, \ldots, T_m$ of T. The *information gain* of attribute A wrt. T is defined as follows:

$$information\ gain(T, A) = entropy(T) - \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot entropy(T_i)$$

Attribute Selection: Example (Information Gain)



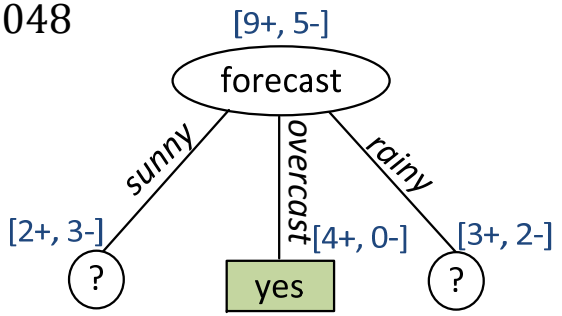
$$information\ gain(T, forecast) = 0.94 - \frac{5}{14} \cdot 0.971 - \frac{4}{14} \cdot 0 - \frac{5}{14} \cdot 0.971 = 0.246$$

$$information\ gain(T, temperature) = 0.94 - \frac{4}{14} \cdot 0.811 - \frac{6}{14} \cdot 0.918 - \frac{4}{14} \cdot 1 = 0.029$$

$$information\ gain(T, humidity) = 0.94 - \frac{7}{14} \cdot 0.985 - \frac{7}{14} \cdot 0.592 = 0.151$$

$$information\ gain(T, wind) = 0.94 - \frac{8}{14} \cdot 0.811 - \frac{6}{14} \cdot 1.0 = 0.048$$

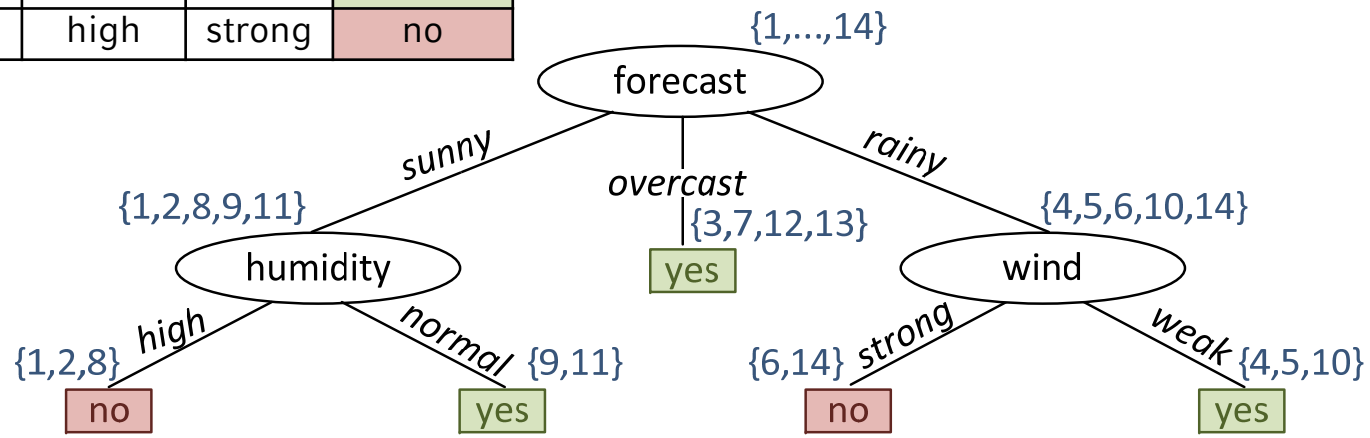
Result: 'forecast' yields the highest information gain



Example: Decision for „playing_tennis“

day	forecast	temperature	humidity	wind	decision
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rainy	mild	high	weak	yes
5	rainy	cool	normal	weak	yes
6	rainy	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rainy	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rainy	mild	high	strong	no

final decision tree



Attribute Selection Measures: Gini Index

- Used in IBM's IntelligentMiner
- The *Gini index* for a set T of training objects is defined as follows

$$gini(T) = 1 - \sum_{j=1}^{k} p_j^2$$

for k classes c_i with frequencies p_i

  - small value of Gini index ⇔ low heterogeneity
  - large value of Gini index ⇔ high heterogeneity

- Let A be the attribute that induced the partitioning $T_1, T_2, \ldots, T_m$ of T. The *Gini index* of attribute A wrt. T is defined as follows:

$$gini_A(T) = \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot gini(T_i)$$

Attribute Selection Measures: Misclassification Error

- The *Misclassification Error* for a set T of training objects is defined as follows

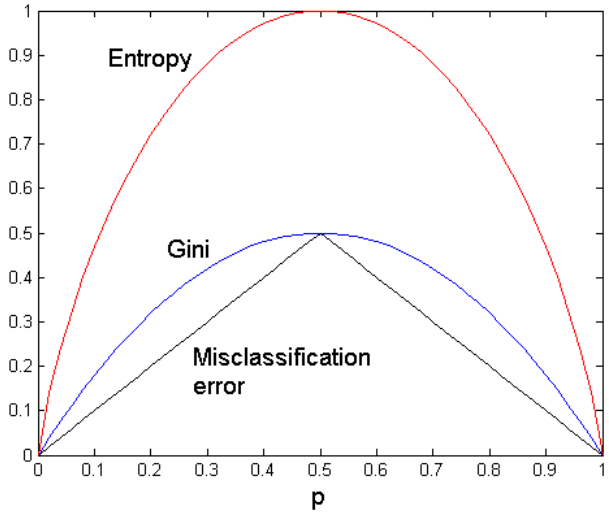
$$Error(T) = 1 - \max_{c_i} p_i$$

for k classes c_i with frequencies p_i

  - small value of Error ⇔ low heterogeneity
  - large value of Error ⇔ high heterogeneity
- Let A be the attribute that induced the partitioning T_1, T_2, …, T_m of T. The *Misclassification Error* of attribute A wrt. T is defined as follows:

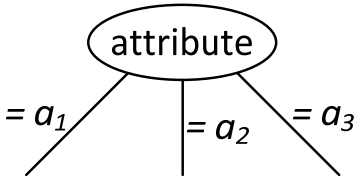
$$Error_A(T) = \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot Error(T_i)$$

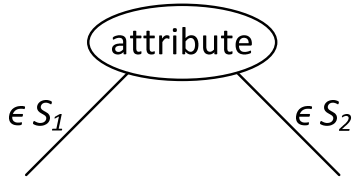
two-class problem:



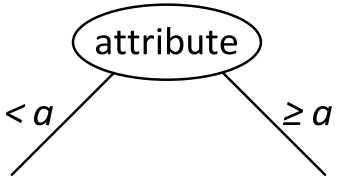
Split Strategies: Types of Splits

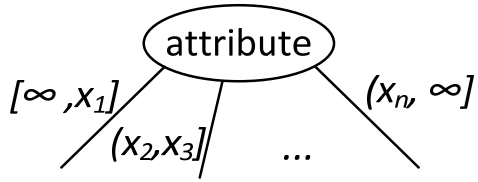
- Categorical attributes
  - split criteria based on equality „*attribute* = *a*“
  - based on subset relationships „*attribute* ∈ *set*“
    → many possible choices (subsets)
    - Choose the best split according to, e.g., gini index





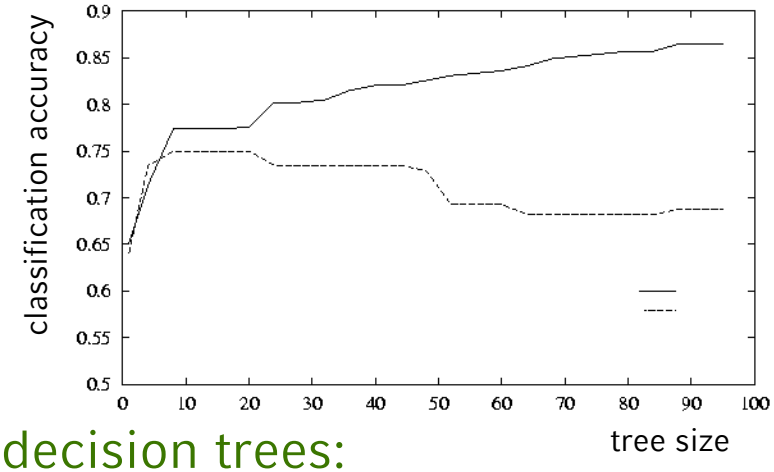
- Numerical attributes
  - split criteria of the form „*attribute* < *a*“
    → many possible choices for the split point
    - One approach: order test samples w.r.t. their attribute value; consider every mean value between two adjacent samples as possible split point; choose best one according to, e.g., gini index
  - Partition the attribute value into a discrete set of intervals (Binning)





Avoid Overfitting in Classification

- The generated tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Result has poor accuracy for unseen samples



- Two approaches to avoid overfitting for decision trees:

1) Post pruning = pruning of overspecialized branches
   - Remove branches from a "fully grown" tree & get a sequence of progressively pruned trees
   - Use a set of data different from the training data to decide which is the "best pruned tree"

Overfitting: Avoidance

2) Prepruning = halt tree construction early, do not split a node if this would result in the goodness measure falling below a threshold
   - Choice of an appropriate value for minimum support
     - *minimum support:* minimum number of data objects a leaf node contains
     - in general, *minimum support* >> 1
   - Choice of an appropriate value for minimum confidence
     - *minimum confidence:* minimum fraction of the majority class in a leaf node
     - typically, minimum confidence << 100%
     - leaf nodes can absorb errors or noise in data records
   - Discussion
     - With Prepruning it is difficult to choose appropriate thresholds
     - Prepruning has less information for the pruning decision than Postpruning. In general, it therefore can be expected to produce decision trees with lower classification quality.
     - Tradeoff: tree construction time ↔ classification quality

Pruning of Decision Trees: Approach Postpruning

Reduced-Error Pruning [Q87]

- Decompose classified data into training set and test set
- Create a decision tree E for the training set
- Prune E by using the test set T
  - determine the interior node v of E whose pruning reduces the number of misclassified data points on T the most (i.e., replace the subtree S with root v by a leaf. Determine the value of the leaf by majority voting)
  - prune
  - finish if no such interior node exists

- only applicable if a sufficient number of classified data is available

[Q87] J.R. Quinlan. *Rule induction with statistical data – a comparison with multiple regression.* In Journal of the Operational Research Society, 38, pages 347-352, 1987.

Pruning of Decision Trees: Approach Postpruning

Minimal Cost Complexity Pruning [BFO+84]

- Does not require a separate test set
  - applicable to small training sets as well
- Pruning of the decision tree by using the training set
  - classification error is no appropriate quality measure
- New quality measure for decision trees:
  - trade-off of classification error and tree size
  - weighted sum of classification error and tree size
- General observation
  - the smaller decision trees yield the better generalization

[BFO+84] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees. Wadsworth International Group*, 1984.

Extracting Classification Rules from Trees

- Represent the knowledge in the form of IF-THEN rules
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction
- The leaf node holds the class prediction
- Rules are easier for humans to understand
- Example

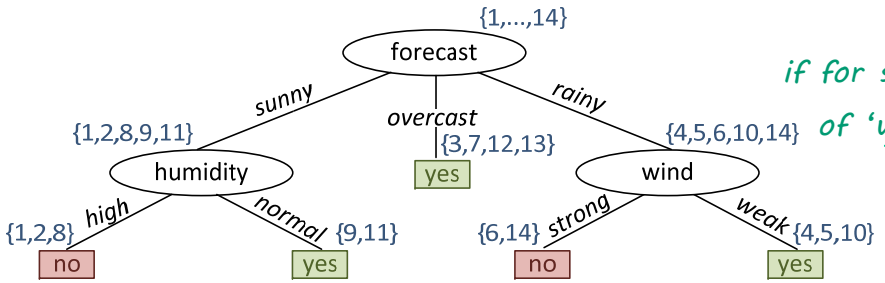
IF forecast = 'overcast'	THEN playing_tennis = 'yes'
IF forecast = 'sunny' AND humidiy = 'high'	THEN playing_tennis = 'no'
IF forecast = 'sunny' AND humidiy = 'normal'	THEN playing_tennis = 'yes'
IF forecast = 'rainy' AND wind = 'strong'	THEN playing_tennis = 'no'
IF forecast = 'rainy' AND wind = 'weak'	THEN playing_tennis = 'yes'

Enhancements to basic decision tree induction

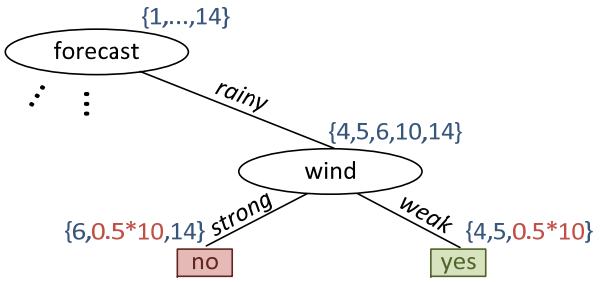
- Handle missing attribute values

  If node *n* tests attribute *A*:

  - assign most common value of *A* among other examples sorted to node *n*
  - assign the most common value of the attribute among other examples with the same target value sorted to node *n*
  - assign probability p_i to each of the possible values v_i of attribute *A* among other examples sorted to node *n*
    - Assign fraction p_i of example to each descendant in tree



if for sample 10 the value of 'wind' is unknown



  - Classify new examples in the same fashion:
    classification decision is the one with the highest probability (sum over all instance fragments of each class decision)

Decision Tree Classifiers: Summary

- Hierarchical linear classifier for data with attributes (categorical or numerical)

- Pro
  - Relatively fast learning speed (in comparison to other classification methods)
  - Fast classification speed
  - Convertible to simple and easy to understand classification rules
  - Often comparable classification accuracy with other classification methods

- Contra
  - Not very stable, small changes of the data can lead to large changes of the tree