Knowledge Discovery in Databases
SS 2016

Chapter 6: Classification

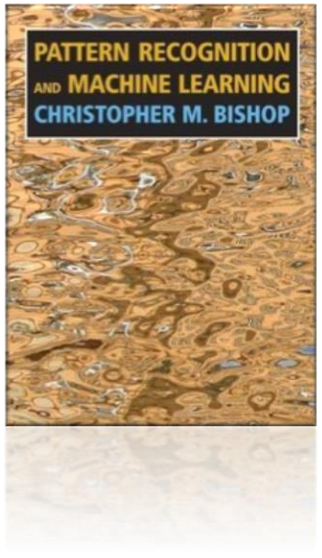
Lecture: Prof. Dr. Thomas Seidl

Tutorials: Julian Busch, Evgeniy Faerman,
Florian Richter, Klaus Schmid

Chapter 5: Classification

Additional literature for this chapter

- Christopher M. Bishop: *Pattern Recognition and Machine Learning*. Springer, Berlin 2006.



Introduction: Example

- Training data

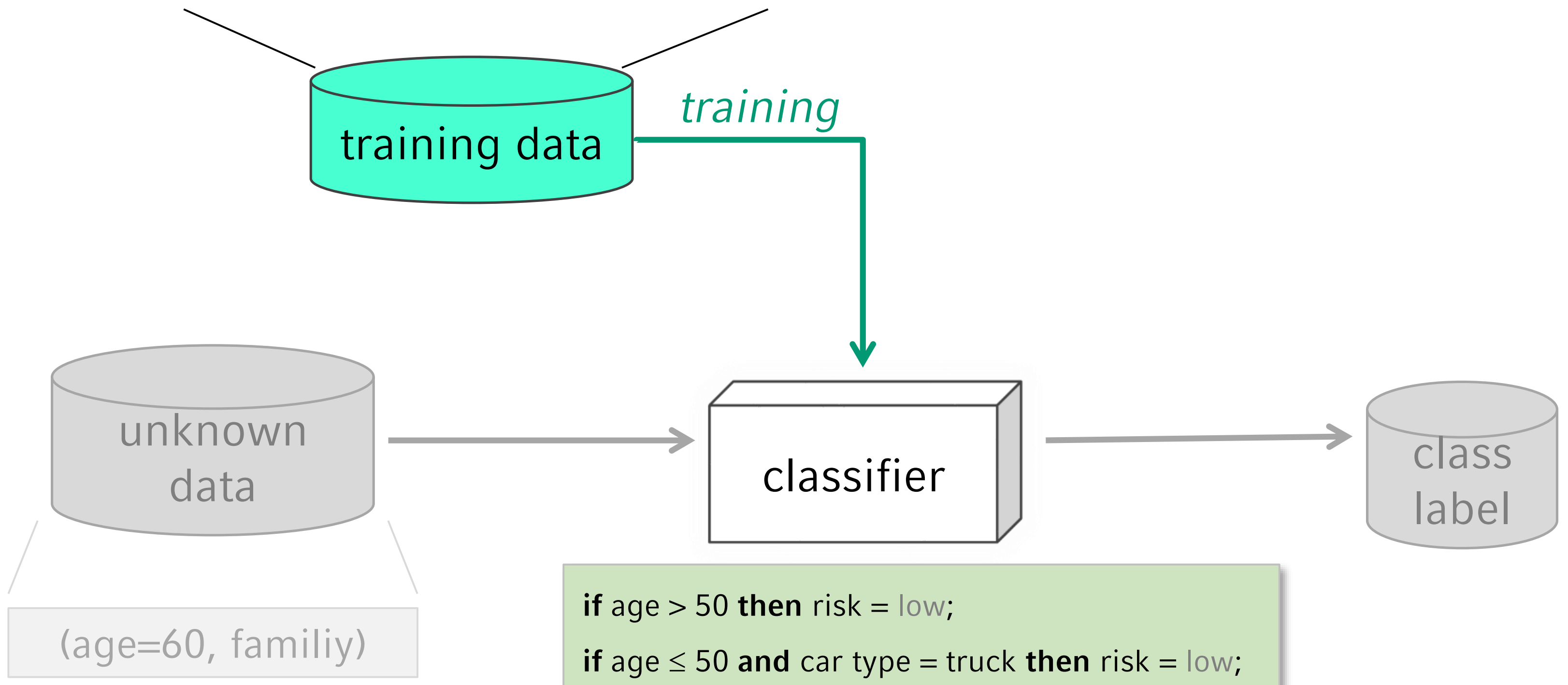
ID	age	car type	risk
1	23	family	high
2	17	sportive	high
3	43	sportive	high
4	68	family	low
5	32	truck	low

- Simple classifier

```
if age > 50 then risk = low;
if age ≤ 50 and car type = truck then risk = low;
if age ≤ 50 and car type ≠ truck then risk = high.
```

Classification: Training Phase (Model Construction)

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



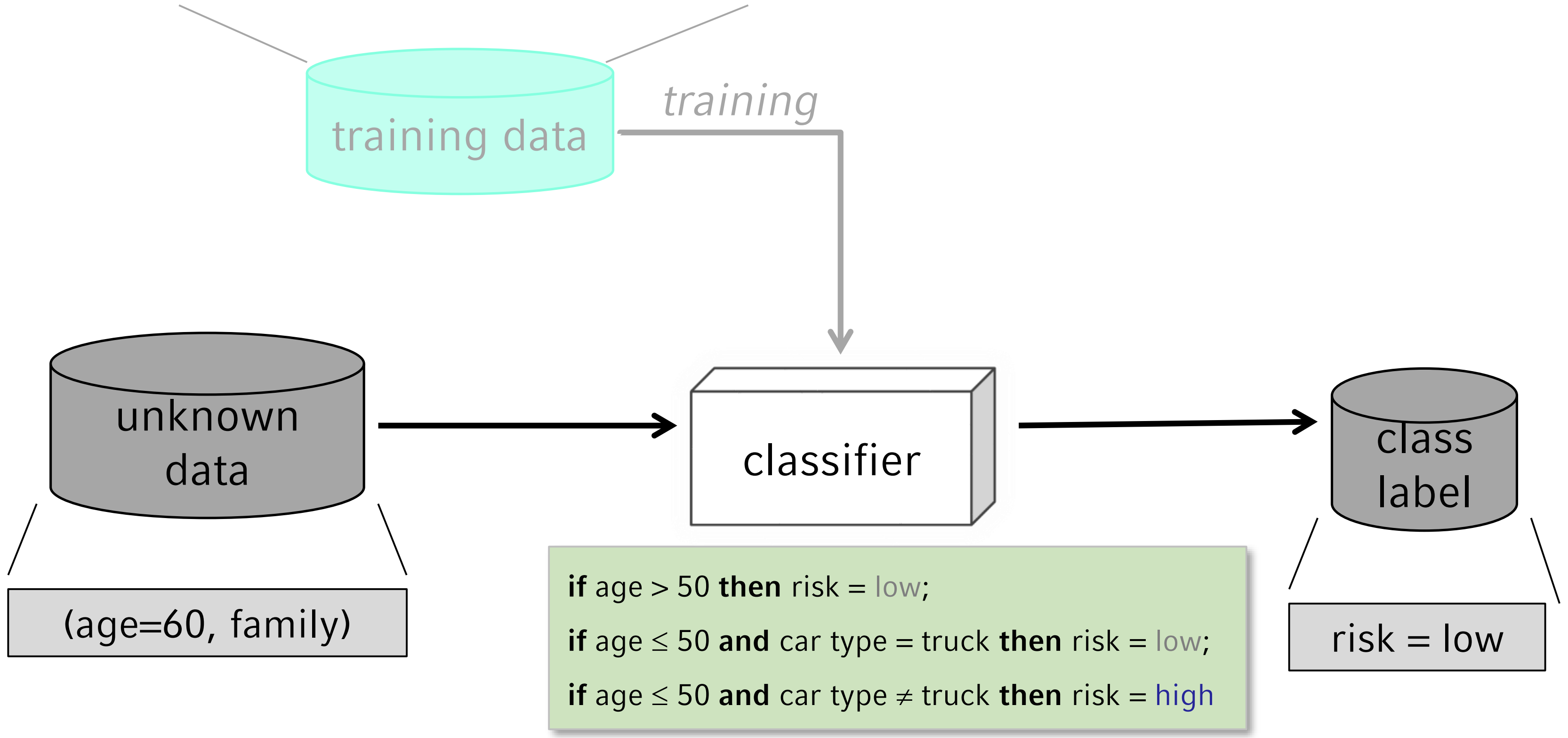
if age > 50 **then** risk = low;

if age ≤ 50 **and** car type = truck **then** risk = low;

if age ≤ 50 **and** car type ≠ truck **then** risk = high

Classification: Prediction Phase (Application)

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



if age > 50 **then** risk = low;

if age ≤ 50 **and** car type = truck **then** risk = low;

if age ≤ 50 **and** car type ≠ truck **then** risk = high

Classification

- The systematic assignment of new observations to known categories according to criteria learned from a training set

- Formally,
  - a classifier K for a model $M(\theta)$ is a function $K_{M(\theta)}\colon D \to Y$, where
    - D: data space
      - *Often d*-dimensional space with attributes a_i, $i = 1, \dots, d$ (not necessarily vector space)
      - Some other space, e.g. metric space
    - $Y = \{y_1, \dots, y_k\}$: set of k distinct class labels y_j, $j = 1, \dots, k$
    - $O \subseteq D$: set of training objects, $o = (o_1, \dots, o_d)$, with known class labels $y \in Y$
  - Classification: application of classifier K on objects from $D - O$
- Model $M(\theta)$ is the "type" of the classifier, and θ are the model parameters
- Supervised learning: find/learn optimal parameters θ for the model $M(\theta)$ from the given training data

Supervised vs. Unsupervised Learning

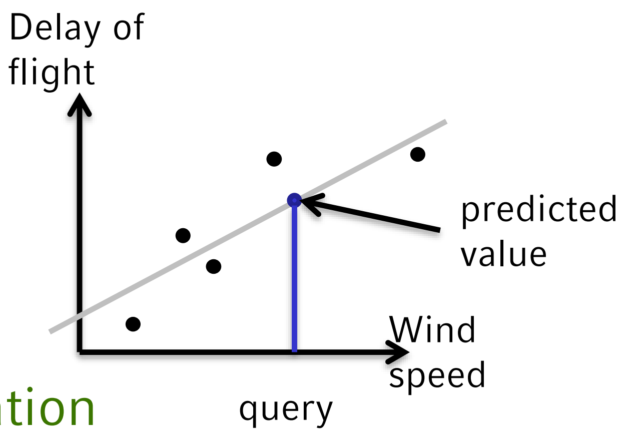
- Unsupervised learning (clustering)
  - The class labels of training data are unknown
  - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data
    - Classes (=clusters) are to be determined

- Supervised learning (classification)
  - Supervision: The training data (observations, measurements, etc.) are accompanied by labels indicating the class of the observations
    - Classes are known in advance (a priori)
  - New data is classified based on information extracted from the training set

[WK91] S. M. Weiss and C. A. Kulikowski. Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems. Morgan Kaufman, 1991.

Numerical Prediction

- Related problem to classification: numerical prediction
  - Determine the numerical value of an object
  - Method: e.g., regression analysis
  - Example: prediction of flight delays

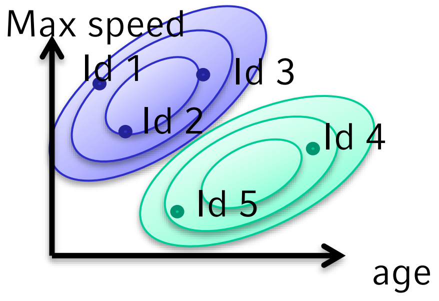


- Numerical prediction is *different* from classification
  - Classification refers to predict categorical class label
  - Numerical prediction models continuous-valued functions
- Numerical prediction is *similar* to classification
  - First, construct a model
  - Second, use model to predict unknown value
    - Major method for numerical prediction is regression
      - Linear and multiple regression
      - Non-linear regression

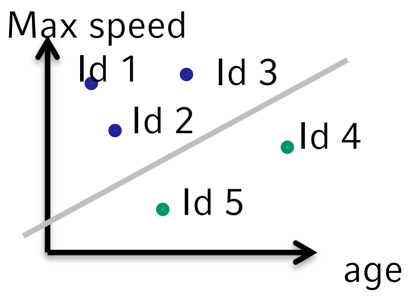
Goals of this lecture

1. Introduction of different classification models

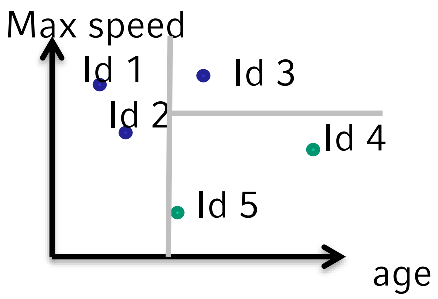
ID	age	car type	Max speed	risk
1	23	family	180	high
2	17	sportive	240	high
3	43	sportive	246	high
4	68	family	173	low
5	32	truck	110	low



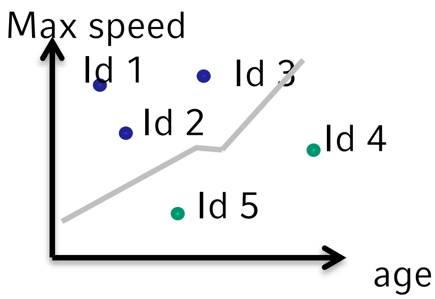
Bayes classifier



Linear discriminant function & SVM



Decision trees



k-nearest neighbor

2. Learning techniques for these models

Quality Measures for Classifiers

- Classification accuracy or classification error (complementary)
- Compactness of the model
  - decision tree size; number of decision rules
- Interpretability of the model
  - Insights and understanding of the data provided by the model
- Efficiency
  - Time to generate the model (training time)
  - Time to apply the model (prediction time)
- Scalability for large databases
  - Efficiency in disk-resident databases
- Robustness
  - Robust against noise or missing values

Evaluation of Classifiers – Notions

- Using training data to build a classifier and to estimate the model's accuracy may result in misleading and overoptimistic estimates
  - due to overspecialization of the learning model to the training data
- *Train-and-Test*: Decomposition of labeled data set O into two partitions
  - *Training data* is used to train the classifier
    - construction of the model by using information about the class labels
  - *Test data* is used to evaluate the classifier
    - temporarily hide class labels, predict them anew and compare results with original class labels
- Train-and-Test is not applicable if the set of objects for which the class label is known is very small

Evaluation of Classifiers – Cross Validation

- *m*-fold *Cross Validation*
  - Decompose data set evenly into *m* subsets of (nearly) equal size
  - Iteratively use $m-1$ partitions as training data and the remaining single partition as test data.
  - Combine the *m* classification accuracy values to an overall classification accuracy, and combine the *m* generated models to an overall model for the data.
- *Leave-one-out* is a special case of cross validation (m=n)
  - For each of the objects o in the data set O:
    - Use set $O \setminus \{o\}$ as training set
    - Use the singleton set $\{o\}$ as test set
  - Compute classification accuracy by dividing the number of correct predictions through the database size $|O|$
  - Particularly well applicable to nearest-neighbor classifiers

Quality Measures: Accuracy and Error

- Let K be a classifier
- Let $C(o)$ denote the correct class label of an object o
- Measure the quality of K:
  - Predict the class label for each object o from a data set $T \subseteq O$
  - Determine the fraction of correctly predicted class labels
  - *Classification Accuracy* of K:

$$G_T(K) = \frac{|\{o \in T, K(o) = C(o)\}|}{|T|}$$

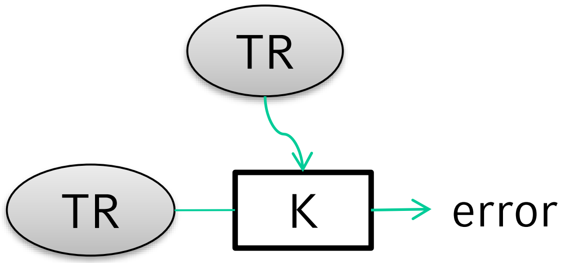
  - *Classification Error* of *K:*

$$F_T(K) = \frac{|\{o \in T, K(o) \neq C(o)\}|}{|T|}$$

Quality Measures: Accuracy and Error

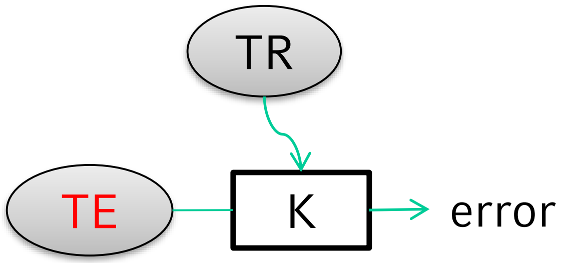
- Let K be a classifier
- Let $TR \subseteq O$ be the training set – used to build the classifier
- Let $TE \subseteq O$ be the test set – used to test the classifier
  - *resubstitution error* of K:

$$F_{TR}(K) = \frac{|\{o \in TR, K(o) \neq C(o)\}|}{|TR|}$$



  - (true) *classification error* of K:

$$F_{TE}(K) = \frac{|\{o \in TE, K(o) \neq C(o)\}|}{|TE|}$$



Quality Measures: Confusion Matrix

- Results on the test set: confusion matrix

classified as ...

correct class label ...	class1	class 2	class 3	class 4	other
class 1	35	1	1	1	4
class 2	0	31	1	1	5
class 3	3	1	50	1	2
class 4	1	0	1	10	2
other	3	1	9	15	13

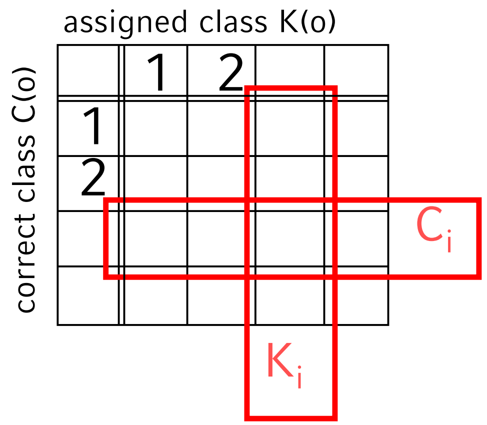
correctly classified objects

- Based on the confusion matrix, we can compute several accuracy measures, including:
  - Classification Accuracy, Classification Error
  - Precision and Recall.

Quality Measures: Precision and Recall

- *Recall:* fraction of test objects of class *i*, which have been identified correctly
- Let $C_i = \{o \in TE \mid C(o) = i\}$, then

$$\text{Recall}_{TE}(K,i) = \frac{|\{o \in C_i \mid K(o) = C(o)\}|}{|C_i|}$$

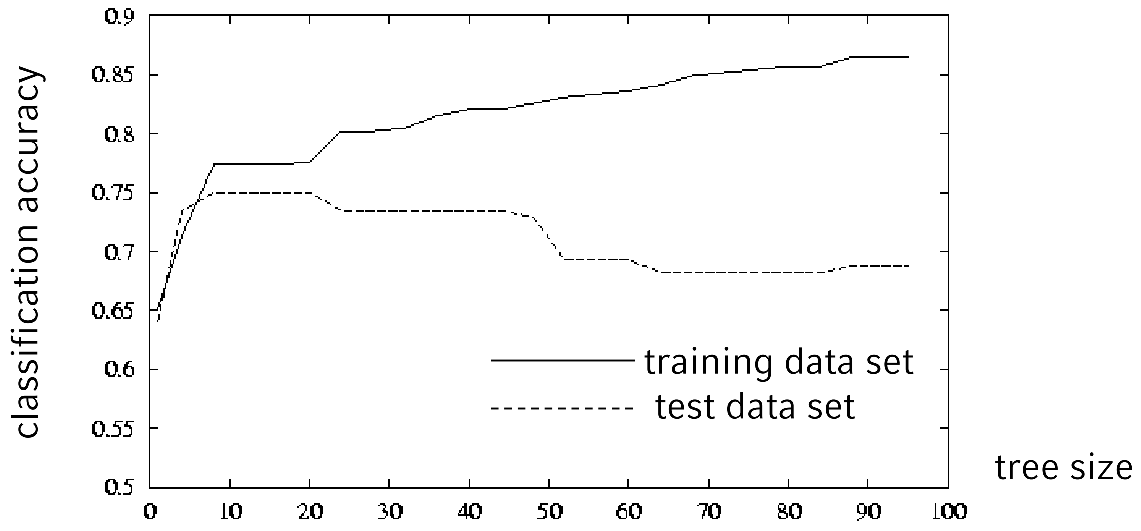


- *Precision:* fraction of test objects assigned to class *i*, which have been identified correctly
- Let $K_i = \{o \in TE \mid K(o) = i\}$, then

$$\text{Precision}_{TE}(K,i) = \frac{|\{o \in K_i \mid K(o) = C(o)\}|}{|K_i|}$$

Overfitting

- Characterization of overfitting:
  There are two classifiers *K* and *K´* for which the following holds:
  - on the training set, *K* has a smaller error rate than *K´*
  - on the overall test data set, *K´* has a smaller error rate than *K*
- Example: Decision Tree



generalization ← *classifier* → specialization
"overfitting"

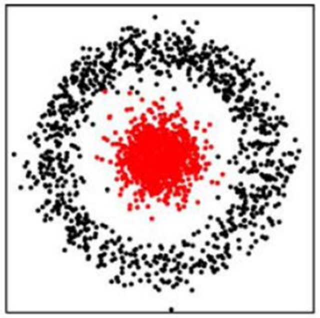
Overfitting (2)

- *Overfitting*
  - occurs when the classifier is too optimized to the (noisy) training data
  - As a result, the classifier yields worse results on the test data set
  - Potential reasons
    - bad quality of training data (noise, missing values, wrong values)
    - different statistical characteristics of training data and test data

- *Overfitting* avoidance
  - Removal of *noisy* and *erroneous* training data; in particular, remove contradicting training data
  - Choice of an appropriate *size* of the training set: not too small, not too large
  - Choice of appropriate sample: sample should describe all aspects of the domain and not only parts of it

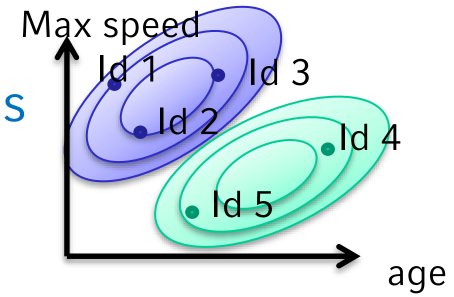
Underfitting

- *Underfitting*
  - Occurs when the classifiers model is too simple, e.g. trying to separate classes linearly that can only be separated by a quadratic surface
  - happens seldomly



- *Trade-off*
  - Usually one has to find a good balance between over- and underfitting

Chapter 6: Classification



Bayes Classification

- Probability based classification
  - Based on likelihood of observed data, estimate explicit probabilities for classes
  - Classify objects depending on costs for possible decisions and the probabilities for the classes
- Incremental
  - Likelihood functions built up from classified data
  - Each training example can incrementally increase/decrease the probability that a hypothesis (class) is correct
  - Prior knowledge can be combined with observed data.
- Good classification results in many applications

Bayes' theorem

- Probability theory:
  - Conditional probability: $P(A|B) = \frac{P(A \wedge B)}{P(B)}$ ("probability of A given B")
  - Product rule: $P(A \wedge B) = P(A|B) \cdot P(B)$
- Bayes' theorem
  - $P(A \wedge B) = P(A|B) \cdot P(B)$
  - $P(B \wedge A) = P(B|A) \cdot P(A)$
  - Since

$$P(A \wedge B) = P(B \wedge A) \Rightarrow$$
$$P(A|B) \cdot P(B) = P(B|A) \cdot P(A) \Rightarrow$$

Bayes' theorem

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Bayes Classifier

- Bayes rule: $p(c_j|o) = \frac{p(o|c_j) \cdot p(c_j)}{p(o)}$

$$\underset{c_j \in C}{\operatorname{argmax}}\{p(c_j|o)\} = \underset{c_j \in C}{\operatorname{argmax}}\left\{\frac{p(o|c_j) \cdot p(c_j)}{p(o)}\right\} = \underset{c_j \in C}{\operatorname{argmax}}\{p(o|c_j) \cdot p(c_j)\}$$

Value of $p(o)$ is constant and does not change the result.

- Final decision rule for the *Bayes classifier*

$$K(o) = c_{max} = \underset{c_j \in C}{\operatorname{argmax}}\{P(o|c_j) \cdot P(c_j)\}$$

- Estimate the apriori probabilities $p(c_j)$ of classes c_j by using the observed frequency of the individual class labels c_j in the training set, i.e., $p(c_j) = \frac{N_{c_j}}{N}$
- How to estimate the values of $p(o|c_j)$?

Density estimation techniques

- Given a database DB, how to estimate conditional probability $p(o|c_j)$?
  - Parametric methods: e.g. single Gaussian distribution
    - Compute by maximum likelihood estimators (MLE), etc.
  - Non-parametric methods: Kernel methods
    - Parzen's window, Gaussian kernels, etc.
  - Mixture models: e.g. mixture of Gaussians (GMM = Gaussian Mixture Model)
    - Compute by e.g. EM algorithm
- Curse of dimensionality often lead to problems in high dimensional data
  - Density functions become too uninformative
  - Solution:
    - Dimensionality reduction
    - Usage of statistical independence of single attributes (extreme case: naïve Bayes)

Naïve Bayes Classifier (1)

- Assumptions of the naïve Bayes classifier
  - Objects are given as *d*-dim. vectors, $o = (o_1, \dots, o_d)$
  - For any given class c_j the attribute values o_i are *conditionally independent*, i.e.

$$p(o_1, \dots, o_d | c_j) = \prod_{i=1}^{d} p(o_i|c_j) = p(o_1|c_j) \cdot \dots \cdot p(o_d|c_j)$$

- Decision rule for the *naïve Bayes classifier*

$$K_{naive}(o) = \underset{c_j \in C}{\operatorname{argmax}} \left\{ p(c_j) \cdot \prod_{i=1}^{d} p(o_i|c_j) \right\}$$

Naïve Bayes Classifier (2)

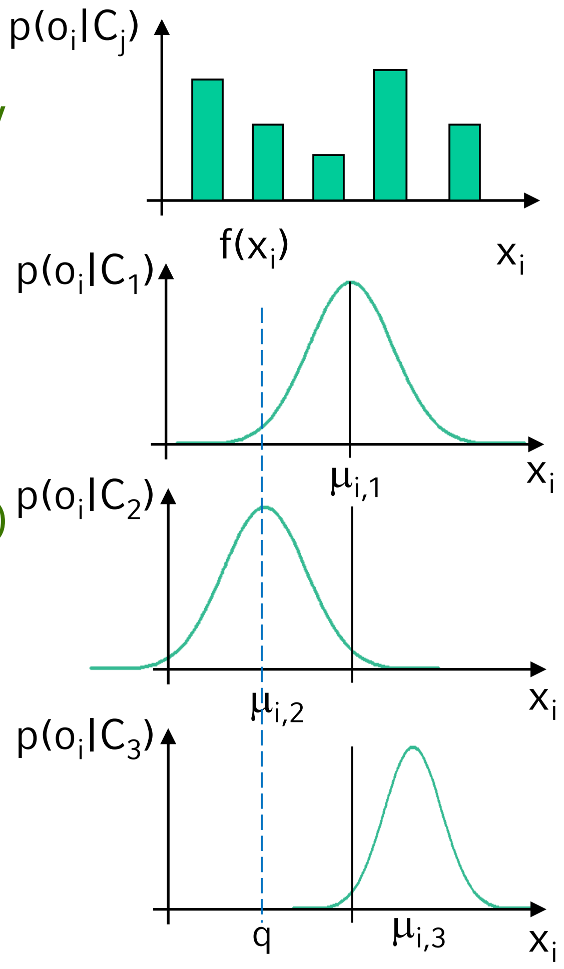
- Independency assumption: $p(o_1, \dots, o_d | c_j) = \prod_{i=1}^{d} p(o_i | c_j)$

- If i-th attribute is categorical:
  $p(o_i|C)$ can be estimated as the relative frequency of samples having value x_i as i-th attribute in class C in the training set

- If i-th attribute is continuous:
  $p(o_i|C)$ can, for example, be estimated through:
  - Gaussian density function determined by $(\mu_{i,j}, \sigma_{i,j})$

  $$\rightarrow\ p(o_i|C_j) = \frac{1}{\sqrt{2\pi}\sigma_{i,j}} e^{-\frac{1}{2}\left(\frac{o_i-\mu_{i,j}}{\sigma_{i,j}}\right)^2}$$

- Computationally easy in both cases



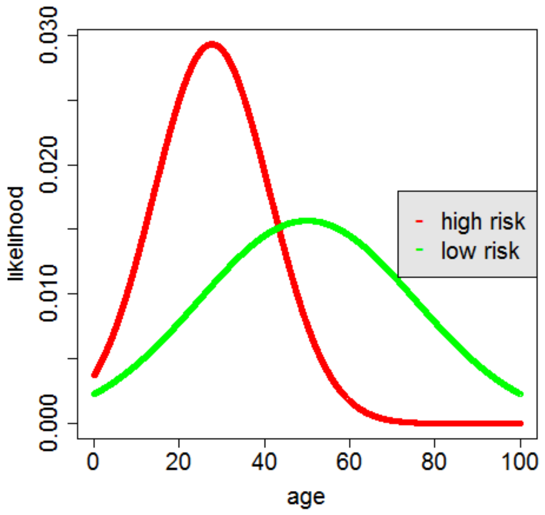
Example: Naïve Bayes Classifier

- Model setup:
  - Age ~ $N(\mu, \sigma)$ (normal distribution)
  - Car type ~ relative frequencies
  - Max speed ~ $N(\mu, \sigma)$ (normal distribution)

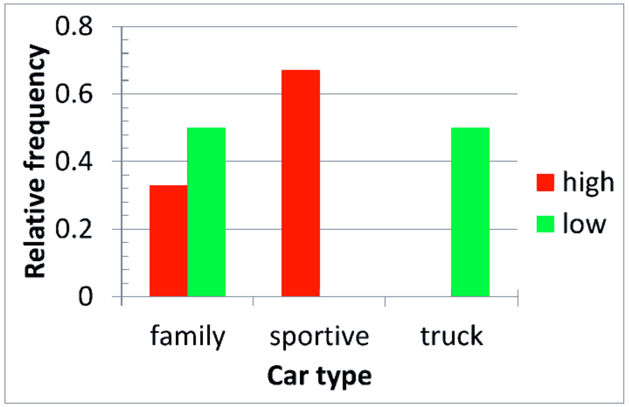
ID	age	car type	Max speed	risk
1	23	family	180	high
2	17	sportive	240	high
3	43	sportive	246	high
4	68	family	173	low
5	32	truck	110	low

Age:

$$\mu_{age}^{high} = 27.67, \sigma_{age}^{high} = 13.61$$
$$\mu_{age}^{low} = 50, \sigma_{age}^{low} = 25.45$$

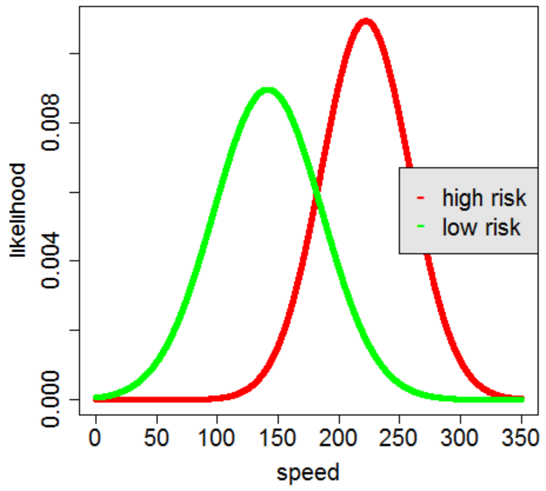


Car type:



Max speed:

$$\mu_{speed}^{high} = 222, \sigma_{speed}^{high} = 36.49$$
$$\mu_{speed}^{low} = 141.5, \sigma_{speed}^{low} = 44.54$$



Example: Naïve Bayes Classifier (2)

- Query: q = (age = 60, car type = family, max speed = 190)
- Calculate the probabilities for both classes:

With:
$$1 = p(high|q) + p(low|q)$$

$$p(high|q) = \frac{p(q|high) \cdot p(high)}{p(q)}$$
$$= \frac{p(age = 60|high) \cdot p(car\ type = family|high) \cdot p(\max speed = 190|high) \cdot p(high)}{p(q)}$$
$$= \frac{N(27.67, 13.61|60) \cdot \frac{1}{3} \cdot N(222,36.49|190) \cdot \frac{3}{5}}{p(q)} = 15.32\%$$

$$p(low|q) = \frac{p(q|low) \cdot p(low)}{p(q)}$$
$$= \frac{p(age = 60|low) \cdot p(car\ type = family|low) \cdot p(\max speed = 190|low) \cdot p(low)}{p(q)}$$
$$= \frac{N(50, 25.45|60) \cdot \frac{1}{2} \cdot N(141.5,44.54|190) \cdot \frac{2}{5}}{p(q)} = 84{,}68\% \quad \longleftarrow$$

Classifier decision

Bayesian Classifier

- Assuming dimensions of $o = (o_1 \ldots o_d)$ are not independent
- Assume multivariate normal distribution (=Gaussian)

$$P(o|C_j) = \frac{1}{(2\pi)^{d/2} |\Sigma_j|^{1/2}} e^{-\frac{1}{2}(o-\mu_j)\Sigma_j^{-1}(o-\mu_j)^T}$$

with

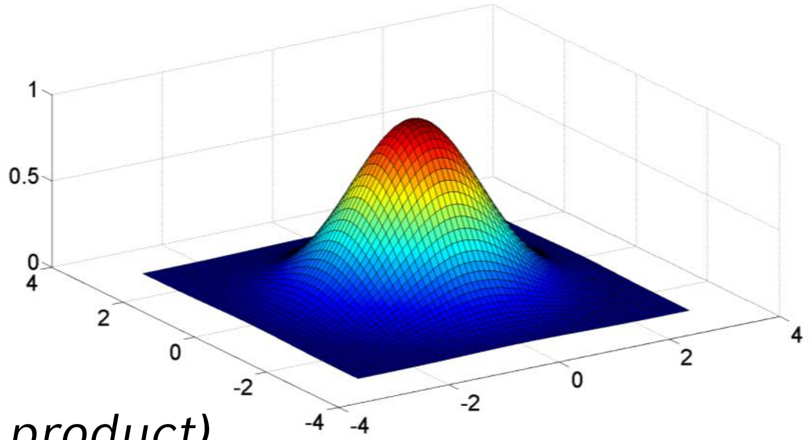
μ_j mean vector of class C_j

N_j is number of objects of class C_j

Σ_j is the $d \times d$ covariance matrix

$$\Sigma_j = \frac{1}{N_j - 1} \sum_{i=1}^{N_j} (o_i - \mu_j)^T \cdot (o_i - \mu_j)$$

(outer product)

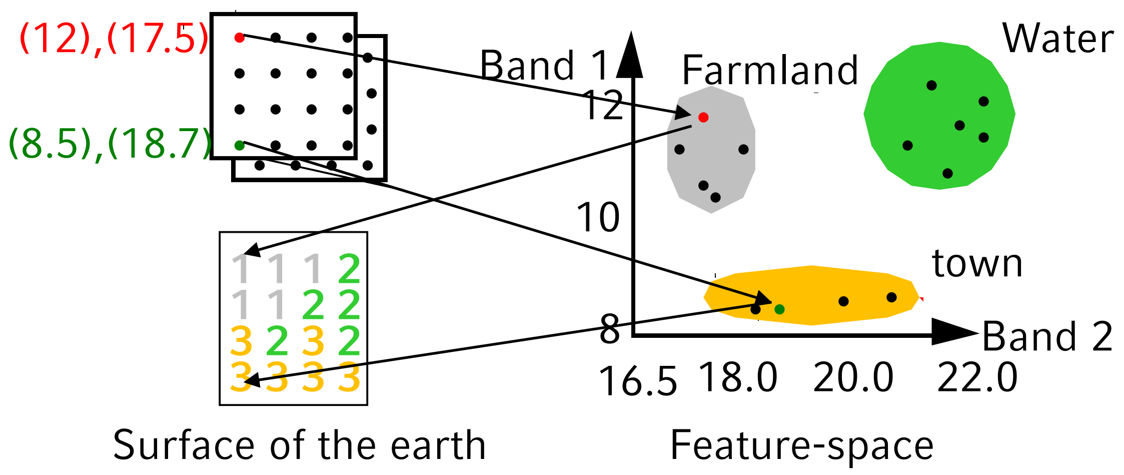


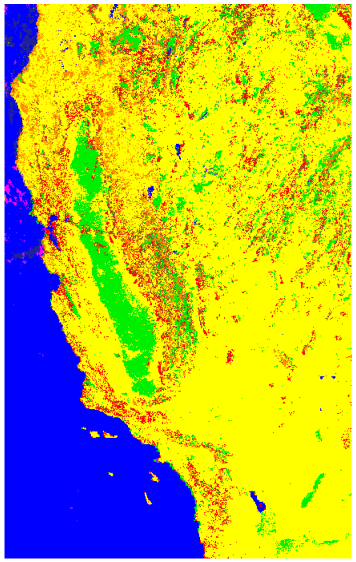
$|\Sigma_j|$ is the determinant of Σ_j and Σ_j^{-1} the inverse of Σ_j

DATABASE SYSTEMS GROUP

Example: Interpretation of Raster Images

- Scenario: automated interpretation of raster images
  - Take an image from a certain region (in *d* different frequency bands, e.g., infrared, etc.)
  - Represent each pixel by *d* values: $(o_1, \ldots, o_d)$
- Basic assumption: different surface properties of the earth („landuse“) follow a characteristic reflection and emission pattern

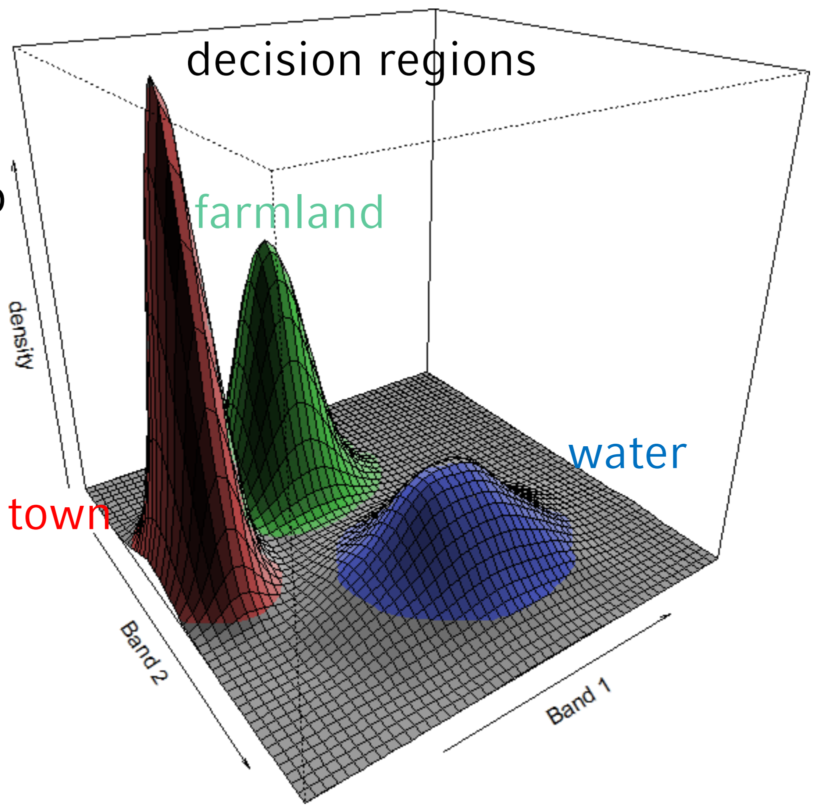




Example: Interpretation of Raster Images

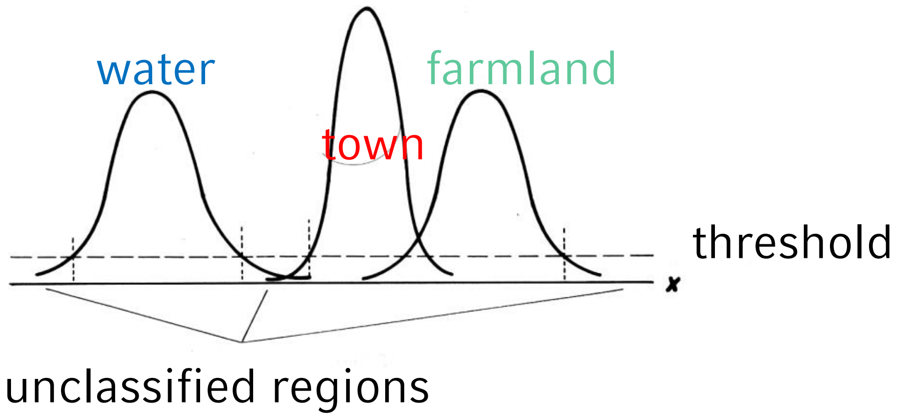
- Application of the Bayes classifier
  - Estimation of the p(*o* | *c*) without assumption of conditional independence
  - Assumption of d-dimensional normal (= Gaussian) distributions for the value vectors of a class

Probability *p* of class membership



Example: Interpretation of Raster Images

- Method: Estimate the following measures from training data
  - μ_j: d-dimensional mean vector of all feature vectors of class C_j
  - Σ_j: $d \times d$ covariance matrix of class C_j
- Problems with the decision rule
  - if likelihood of respective class is very low
  - if several classes share the same likelihood



Bayesian Classifiers – Discussion

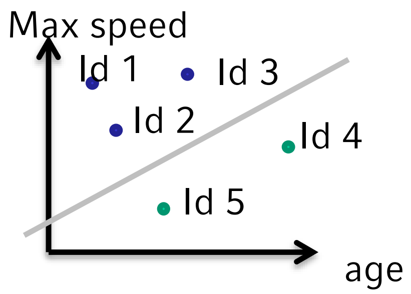
- Pro
  - High classification accuracy for many applications if density function defined properly
  - Incremental computation
    → many models can be adopted to new training objects by updating densities
    - For Gaussian: store *count*, *sum*, *squared sum* to derive *mean*, *variance*
    - For histogram: store *count* to derive *relative frequencies*
  - Incorporation of expert knowledge about the application in the prior $P(C_i)$

- Contra
  - Limited applicability
    → often, required conditional probabilities are not available
  - Lack of efficient computation
    → in case of a high number of attributes
    → particularly for Bayesian belief networks

The independence hypothesis…

- … makes efficient computation possible
- … yields optimal classifiers when satisfied
- … but is seldom satisfied in practice, as attributes (variables) are often correlated.
- Attempts to overcome this limitation:
  - Bayesian networks, that combine Bayesian reasoning with causal relationships between attributes
  - Decision trees, that reason on one attribute at the time, considering most important attributes first

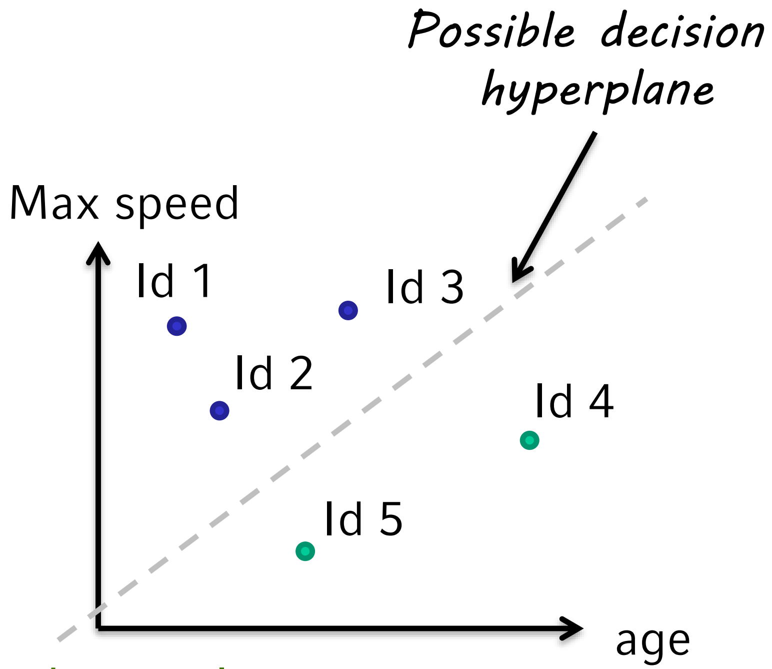
Chapter 6: Classification



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?

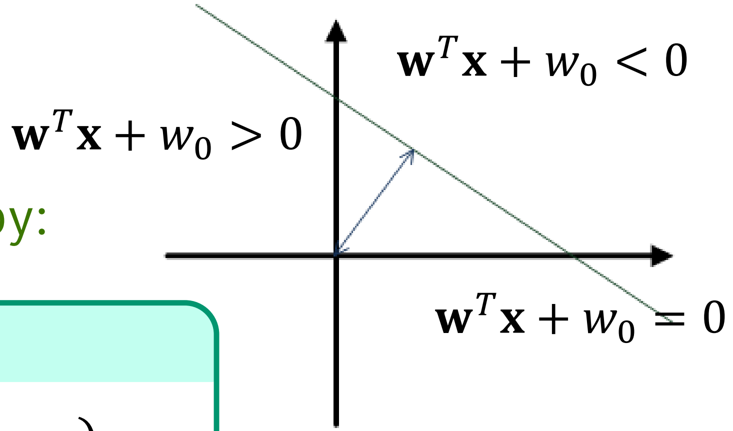
Basic notions

- Recall some general algebraic notions for a vector space V:
  - $\langle \mathbf{x}, \mathbf{y} \rangle$ denotes an inner product of two vectors $\mathbf{x}, \mathbf{y} \in V$:
    e.g., the scalar product: $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y} = \sum_{i=1}^{d} (\mathrm{x}_i \cdot \mathrm{y}_i)$
  - $H(\mathbf{w}, w_0)$ denotes a hyperplane with normal vector $\mathbf{w}$ and constant term w_0:
    $\mathbf{x} \in H(\mathbf{w}, w_0) \Leftrightarrow \langle \mathbf{w}, \mathbf{x} \rangle + w_0 = 0$
  - The normal vector $\mathbf{w}$ may be normalized to $\boldsymbol{w}'$:
    $$\mathbf{w}' = \frac{1}{\sqrt{\langle \mathbf{w}, \mathbf{w} \rangle}} \cdot \mathbf{w} \;\Rightarrow \langle \mathbf{w}', \mathbf{w}' \rangle = 1$$
  - Distance of a vector x to the hyperplane $H(\mathbf{w}', w_0)$:
    $dist\big(\mathbf{x}, H(\mathbf{w}', w_0)\big) = |\langle \mathbf{w}', \mathbf{x} \rangle + w_0|$

Formalization

- Consider a two-class example (generalizations later on):
  - D: d-dimensional vector space with attributes a_i, $i = 1, \ldots, d$
  - $Y = \{-1, 1\}$ set of 2 distinct class labels y_j
  - $O \subseteq D$: set of objects, $\mathbf{o} = (o_1, \ldots, o_d)$, with known class labels $y \in Y$ and cardinality of $|O| = N$
- A hyperplane $H(\mathbf{w}, w_0)$ with normal vector $\mathbf{w}$ and constant term w_0

$$\mathbf{x} \in H \Leftrightarrow \mathbf{w}^T\mathbf{x} + w_0 = 0$$


$\mathbf{w}^T\mathbf{x} + w_0 < 0$
$\mathbf{w}^T\mathbf{x} + w_0 > 0$
$\mathbf{w}^T\mathbf{x} + w_0 = 0$

- Classification rule (linear classifier) given by:

Classification rule

$$K_{H(\mathbf{w},w_0)}(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + w_0)$$

Optimal parameter estimation

- How to estimate optimal parameters $\mathbf{w}, w_0$?
  1. Define an objective/loss function $L(\cdot)$ that assigns a value (e.g. the error on the training set) to each parameter-configuration
  2. Optimal parameters minimize/maximize the objective function

- How does an objective function look like?
  - Different choices possible
  - Most intuitive: each misclassified object contributes a constant (loss) value → 0-1 loss

0-1 loss objective function for linear classifier

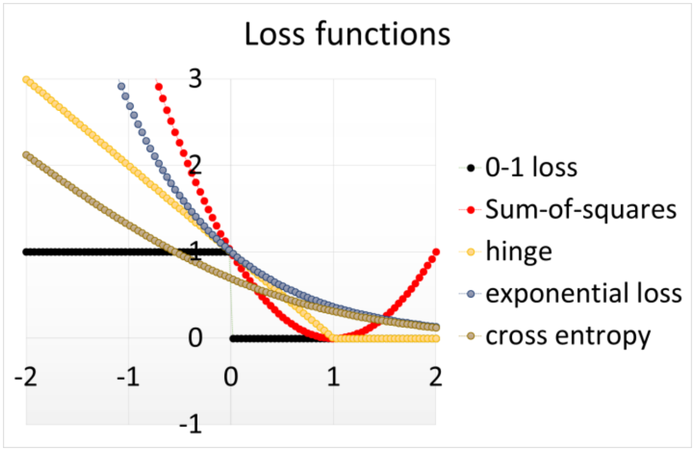
$$L(\mathbf{w}, w_0) = \min_{\mathbf{w}, w_0} \sum_{n=1}^{N} I(y_i \neq K_{H(\mathbf{w}, w_0)}(\mathbf{x}_i))$$

where $I(condition) = 1$, if condition holds, 0 otherwise

DATABASE SYSTEMS GROUP

Loss functions

- 0-1 loss
  - Minimize the overall number of training errors, but…
    - NP-hard to optimize in general (non-smooth, non-convex)
    - Small changes of $\mathbf{w}, w_0$ can lead to large changes of the loss
- Alternative convex loss functions
  - Sum-of-squares loss: $(\mathbf{w}^T\mathbf{x}_i + w_0 - y_i)^2$
  - Hinge loss: $(1 - y_i(w_0 + \mathbf{w}^T\mathbf{x}_i))_+ = \max\{0,\ 1 - y_i(w_0 + \mathbf{w}^T\mathbf{x}_i\}$ (SVM)
  - Exponential loss: $e^{-y_i(w_0+\mathbf{w}^T\mathbf{x}_i)}$ (AdaBoost)
  - Cross-entropy error: $-y_i \ln g(\mathbf{x}_i) + (1 - y_i)\ln(1 - g(\mathbf{x}_i))$ (Logistic regression)
    where $g(\mathbf{x}) = \frac{1}{1+e^{-(w_0+\mathbf{w}^T\mathbf{x})}}$
  - … and many more
- Optimizing different loss function leads to several classification algorithms
- Next, we derive the optimal parameters for the sum-of-squares loss



Optimal parameters for SSE loss

- Loss/Objective function: sum-of-squares error to real class values

Objective function

$$SSE(\mathbf{w}, w_0) = \sum_{i=1..N} \{(\mathbf{w}^T \mathbf{x}_i + w_0) - y_i\}^2$$

- Minimize the error function for getting optimal parameters
  - Use standard optimization technique:
    1. Calculate first derivative
    2. Set derivative to zero and compute the global minimum (SSE is a convex function)

Optimal parameters for SSE loss (cont'd)

- Transform the problem for simpler computations
  - $w^T o + w_0 = \sum_{i=1}^{d} w_i \cdot o_i + w_0 = \sum_{i=0}^{d} w_i \cdot o_i$, with $o_0 = 1$
  - For $\mathbf{w}$ let $\widetilde{\mathbf{w}} = (w_0, \dots, w_d)^T$
- Combine the values to matrices $\tilde{O} = \begin{pmatrix} 1 & o_{1,1} & \dots & o_{1,d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & o_{N,1} & \dots & o_{N,d} \end{pmatrix}$, $Y = \begin{pmatrix} y_1 \\ \dots \\ y_N \end{pmatrix}$
- Then the sum-of-squares error is equal to:

$$\boxed{\sum_i a_{ii}^2 = \operatorname{tr}(A^T A)}$$

$$SSE(\widetilde{\mathbf{w}}) = \frac{1}{2} \operatorname{tr}\left(\left(\tilde{O}\widetilde{\mathbf{w}} - Y\right)^T \left(\tilde{O}\widetilde{\mathbf{w}} - Y\right)\right)$$

Optimal parameters for SSE loss (cont'd)

- Take the derivative: $\boxed{\frac{\partial}{\partial \tilde{\mathbf{w}}} SSE(\tilde{\mathbf{w}}) = \tilde{O}^T\left(\tilde{O}\tilde{\mathbf{w}} - Y\right)}$

- Solve $\frac{\partial}{\partial \tilde{\mathbf{w}}} SSE(\hat{\mathbf{w}}) = 0$:

$$\tilde{O}^T\left(\tilde{O}\hat{\mathbf{w}} - Y\right) = 0 \Leftrightarrow \tilde{O}\hat{\mathbf{w}} = Y \Leftrightarrow \hat{\mathbf{w}} = \overbrace{\left(\tilde{O}^T\tilde{O}\right)^{-1}\tilde{O}^T}^{\text{Left-inverse of } \tilde{O} \text{ ("Moore-Penrose-Inverse")}} Y$$

- Set $\hat{\mathbf{w}} = \left(\tilde{O}^T\tilde{O}\right)^{-1}\tilde{O}^T Y$

- Classify new point $\mathbf{x}$ with $\mathbf{x}_0 = 1$:

Classification rule

$$K_{H(\hat{\mathbf{w}},w_0)}(\mathbf{x}) = \operatorname{sign}(\hat{\mathbf{w}}^T\mathbf{x})$$

Example SSE

- Data (consider only age and max. speed):

$$\tilde{O} = \begin{pmatrix} 1 & 23 & 180 \\ 1 & 17 & 240 \\ 1 & 43 & 246 \\ 1 & 68 & 173 \\ 1 & 32 & 110 \end{pmatrix}, Y = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}$$

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

encode classes as $\{\text{high} = 1, \text{low} = -1\}$

$$\Rightarrow \left(\tilde{O}^T\tilde{O}\right)^{-1}\tilde{O}^T = \begin{pmatrix} 0.7647 & -0.0678 & -0.9333 & -0.4408 & 1.6773 \\ -0.0089 & -0.0107 & 0.0059 & 0.0192 & -0.0055 \\ -0.0012 & 0.0034 & 0.0048 & -0.0003 & -0.0067 \end{pmatrix}$$

$$\Rightarrow \hat{\mathbf{w}} = \left(\tilde{O}^T\tilde{O}\right)^{-1}\tilde{O}^T Y = \begin{pmatrix} w_0 \\ w_{age} \\ w_{maxspeed} \end{pmatrix} = \begin{pmatrix} -1.4730 \\ -0.0274 \\ 0.0141 \end{pmatrix}$$

Example SSE (cont'd)

- Model parameter:

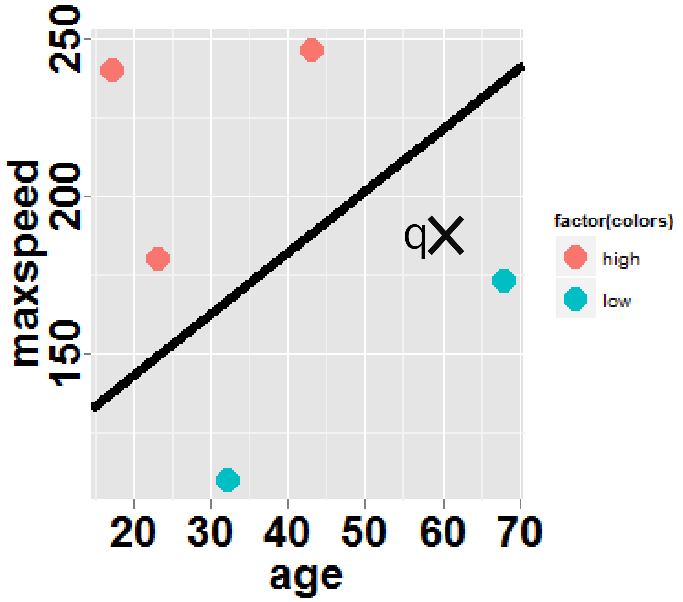
$$\hat{\mathbf{w}} = \left(\tilde{O}^T\tilde{O}\right)^{-1}\tilde{O}^T Y = \begin{pmatrix} w_0 \\ w_{age} \\ w_{maxspeed} \end{pmatrix} = \begin{pmatrix} -1.4730 \\ -0.0274 \\ 0.0141 \end{pmatrix}$$

$$\Rightarrow K_{H(\mathbf{w},w_0)}(\mathbf{x}) = \mathrm{sign}\left(\begin{pmatrix} -0.0274 \\ 0.0141 \end{pmatrix}^T \mathbf{x} - 1.4730\right)$$

Query: q = (age=60, max speed = 190)

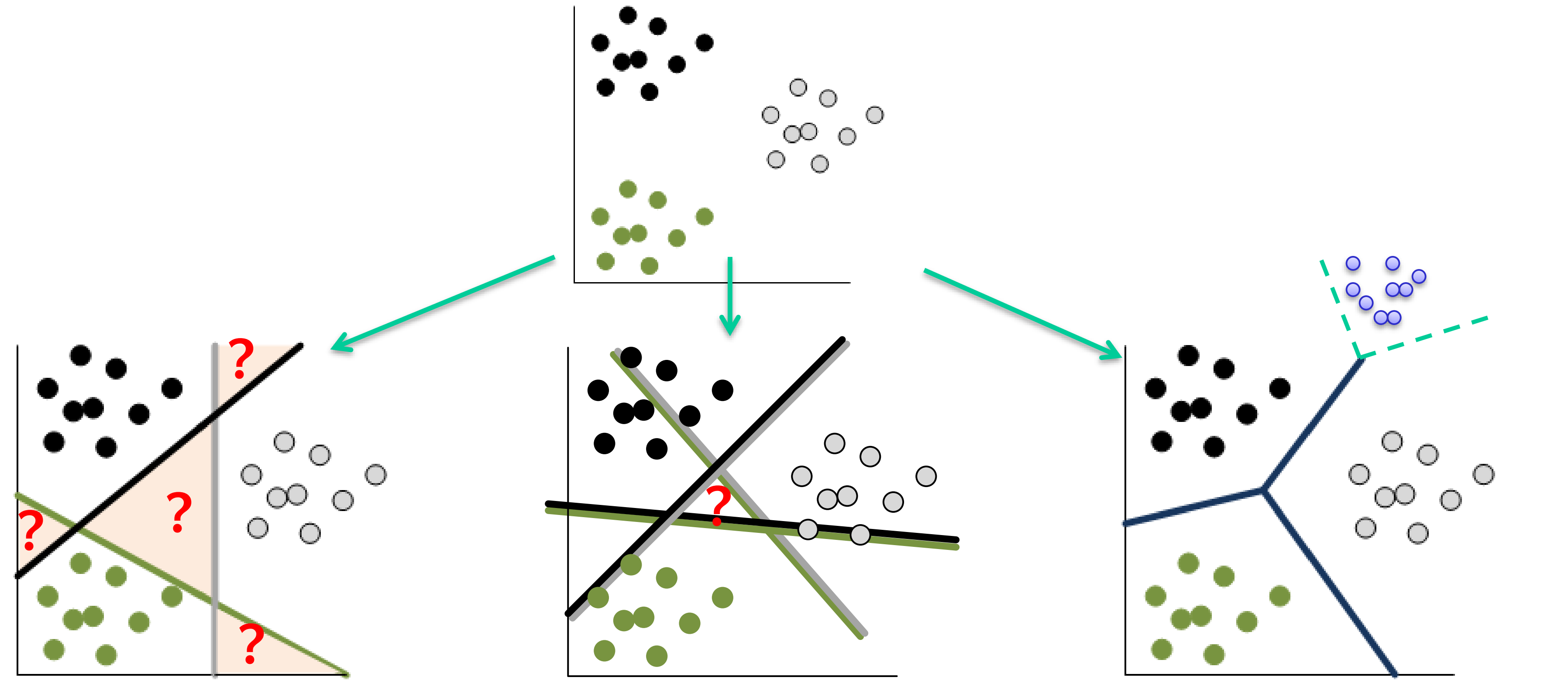
$\Rightarrow \mathrm{sign}(\hat{\mathbf{w}}^T q) = \mathrm{sign}(-0.4397) = -1$

⇒ Class = low



Extension to multiple classes

- Assume we have more than two (k > 2) classes. What to do?



One-vs-the-rest ("one-vs-all") $\rightarrow$ *k classifiers*

One-vs-one (Majority vote of classifiers) $\rightarrow \frac{k(k-1)}{2}$ *classifiers*

Multiclass linear classifier $\rightarrow$ *k classifiers*

Extension to multiple classes (cont'd)

- Idea of multiclass linear classifier
  - Take k linear functions of the form $H_{\mathbf{w_j},w_{j,0}}(\mathbf{x}) = \mathbf{w}_j^T\mathbf{x} + w_{j,0}$
  - Decide for class y_j:

$$\mathrm{y_j} = \arg \max_{j=1,\dots,k} H_{\mathbf{w}_j,w_{j,0}}(\mathbf{x})$$

- Advantage
  - No ambiguous regions except for points on decision hyperplanes

- The optimal parameter estimation is also extendable to k classes $Y = (y_1, \dots, y_k)$

Discussion (SSE)

- Pro
  - Simple approach
  - Closed form solution for parameters
  - Easily extendable to non-linear spaces (later on)

- Contra
  - Sensitive to outliers → not stable classifier
    - How to define and efficiently determine the maximum stable hyperplane?
  - Only good results for linearly separable data
  - Expensive computation of selected hyperplanes

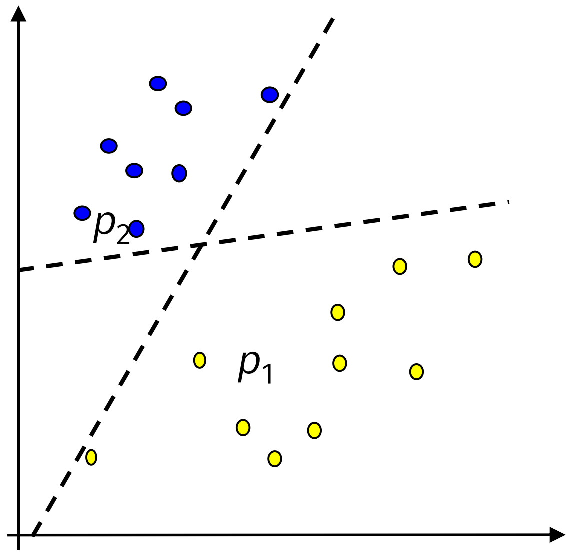
- Approach to solve the problems
  - Support Vector Machines (SVMs) [Vapnik 1979, 1995]

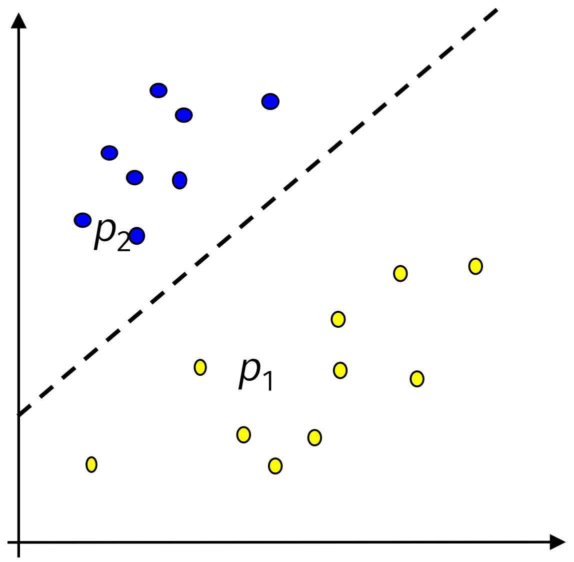
Chapter 6: Classification

DATABASE SYSTEMS GROUP

Maximum Margin Hyperplane

- Question: How to define the notion of the "best" hyperplane differently?
  - Use another objective function that results in the maximum margin hyperplane

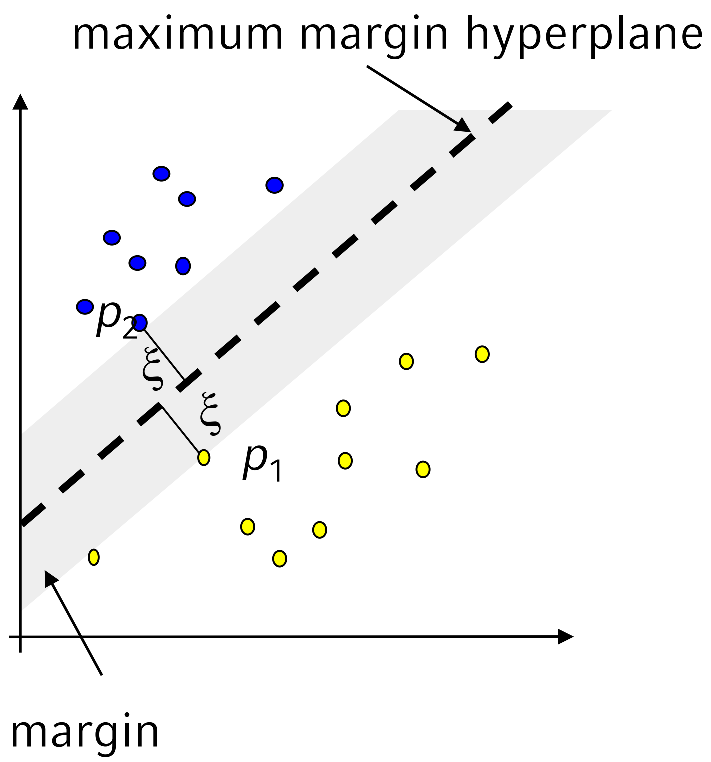




- Criteria
  - Stability at insertion
  - Distance to the objects of both classes

Support Vector Machines: Principle

- Basic idea: Linear separation with the Maximum Margin Hyperplane (MMH)
  - Distance to points from any of the two sets is maximal, i.e., at least ξ
  - Minimal probability that the separating hyperplane has to be moved due to an insertion
  - Best generalization behavior
- MMH is "maximally stable"
- MMH only depends on points p_i whose distance to the hyperplane is exactly ξ
  - p_i is called a *support vector*



Computation of the Maximum Margin Hyperplane

Two assumptions for classifying x_i (class 1: $y_i = +1$, class 2: $y_i = -1$):

1) The classification is accurate (no error)

$$\left.\begin{array}{l} y_i = -1 \;\Rightarrow\; \langle \mathbf{w}, \mathbf{x}_i \rangle + b < 0 \\ y_i = +1 \;\Rightarrow\; \langle \mathbf{w}, \mathbf{x}_i \rangle + b > 0 \end{array}\right\} \;\Leftrightarrow\; y_i \cdot \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) > 0$$

2) The margin is maximal

- Let ξ denote the minimum distance of any training object (TR = training set) x_i to the hyperplane *H(w,b):*

normalized

$$\xi = \min_{x_i \in TR} \left| \langle \overset{\text{normalized}}{\mathbf{w}'}, \mathbf{x}_i \rangle + b \right|$$

- Then: Maximize ξ subject to $\forall i \in \{1, \ldots, n\}$: $\quad y_i \cdot (\langle \mathbf{w}', \mathbf{x}_i \rangle + b) \geq \xi$

DATABASE SYSTEMS GROUP

Maximum Margin Hyperplane

- Maximize ξ subject to $\forall i \in [1..n]$: $y_i \cdot (\langle \mathbf{w}', \mathbf{x}_i \rangle + b) \geq \xi$
- Scaling $\mathbf{w}'$ by $\frac{1}{\xi}$, i.e. $\mathbf{w} = \frac{\mathbf{w}'}{\xi}$ yields the rephrased condition

  $\forall i \in [1..n]$: $y_i \cdot (\langle \mathbf{w}, \mathbf{x}_i \rangle + b') \geq 1$

- *Maximizing* ξ corresponds to *minimizing* $\langle w, w \rangle = \frac{\langle w', w' \rangle}{\xi^2}$:

<u>Primary optimization problem:</u>

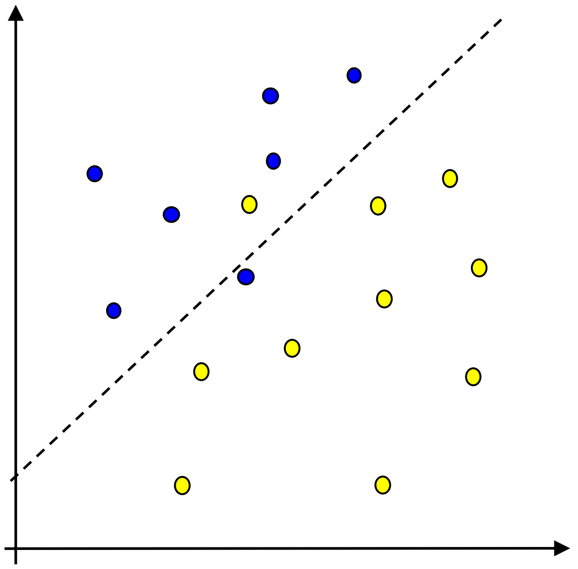
Find a vector $\boldsymbol{w}$ and value b that minimizes $\langle w, w \rangle = |w|^2$

subject to $\forall i \in \{1 \dots n\}$: $y_i \cdot (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$

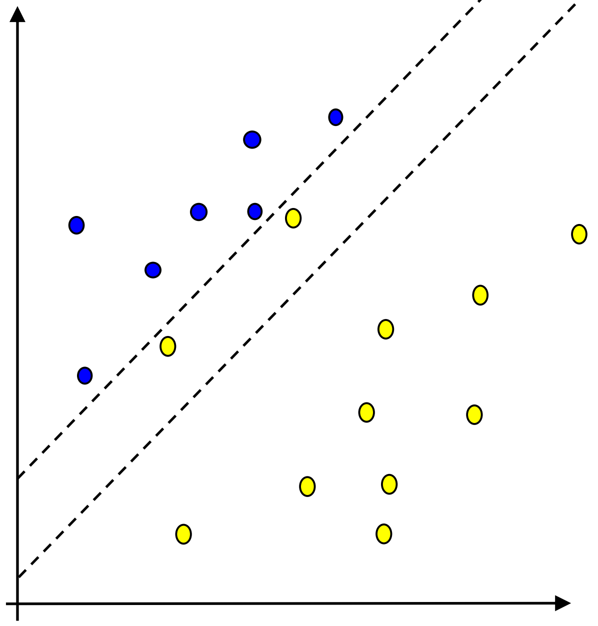
- Convex optimization problem
  - Quadratic programming problem with linear constraints
    → Solution can be obtained by Lagrangian Theory

Soft Margin Optimization

- Problem of Maximum Margin Optimization:
  How to treat non-linearly separable data?
  - Two typical problems:



data points are not separable

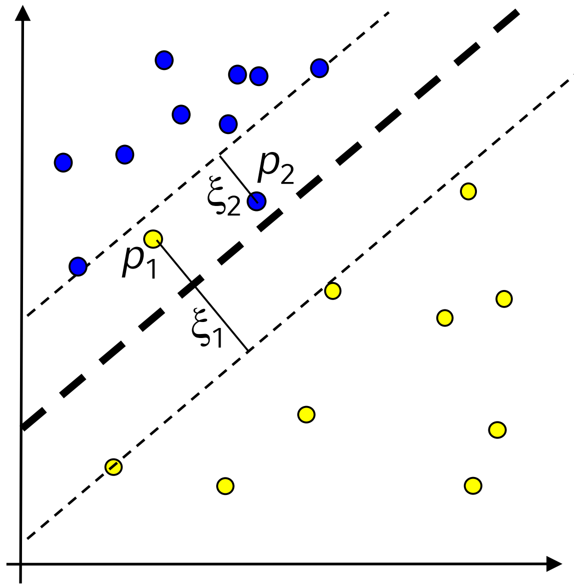


complete separation is not optimal (overfitting)

- Trade-off between training error and size of margin

Soft Margin Optimization

- Additionally regard the number of training errors when optimizing:
  - ξ_i is the distance from p_i to the margin (often called slack variable)
    - $\xi_i = 0$ for points on the correct side
    - $\xi_i > 0$ for points on the wrong side
  - *C* controls the influence of single training vectors



Primary optimization problem with soft margin:

Find an $H(\mathbf{w}, b)$ that minimizes $\frac{1}{2}\langle\mathbf{w},\mathbf{w}\rangle + C \cdot \sum_{i=1}^{n} \xi_i$

subject to $\forall i \in \{1, \dots, n\}$: $y_i \cdot (\langle\mathbf{w}, \mathbf{x}_i\rangle + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$

DATABASE SYSTEMS GROUP

Soft Margin Optimization

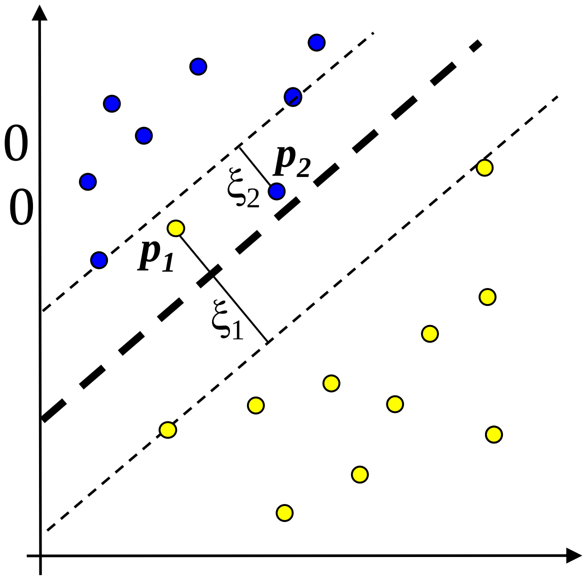
Dual optimization problem with Lagrange multipliers (Wolfe dual):

Dual Optimization Problem:

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

subject to $\sum_{i=1}^{n} \alpha_i \cdot y_i = 0$ and $0 \le \alpha_i \le C$

$\alpha_i = 0$:	p_i is not a support vector
$\alpha_i = C$:	p_i is a support vector with $\xi_i > 0$
$0 < \alpha_i < C$:	p_i is a support vector with $\xi_i = 0$



Decision rule:

SV = support vectors

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

Support Vector Machines: Discussion

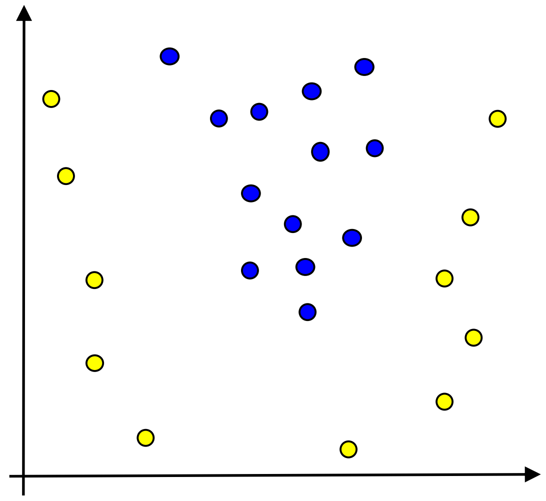
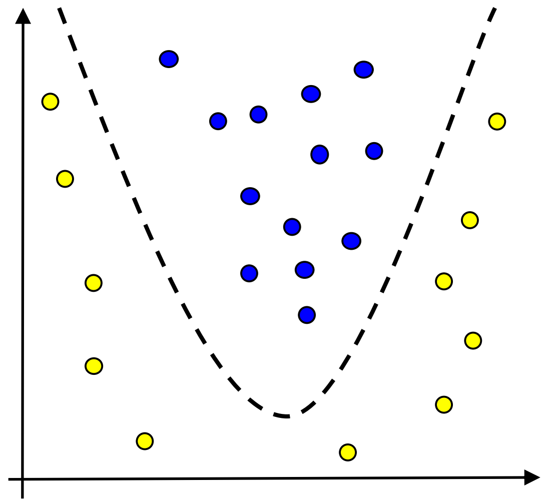
- Pro
  - generate classifiers with a high classification accuracy
  - relatively weak tendency to overfitting (generalization theory)
  - efficient classification of new objects
    - due to often small number of support vectors
  - compact models

- Contra
  - training times may be long (appropriate feature space may be very high-dimensional)
  - expensive implementation

Chapter 6: Classification

Non-Linearly Separable Data Sets

- Problem: For real data sets, a linear separation with a high classification accuracy often is not possible

Example for a quadratically separable data set

- Idea: Transform the data non-linearly into a new space, and try to separate the data in the new space linearly (extension of the hypotheses space)

Extension of the Hypotheses Space

- Principle

$$\begin{tikzcd}\text{input space} \arrow[r, "\phi"] & \text{extended feature space}\end{tikzcd}$$

  - Try to linearly separate in the extended feature space

- Example

$$\begin{tikzcd}(x, y, z) \arrow[r, "\phi"] & (x, y, z, x^2, xy, xz, y^2, yz, z^2)\end{tikzcd}$$

  - Here: a hyperplane in the extended feature space is a polynomial of degree 2 in the input space

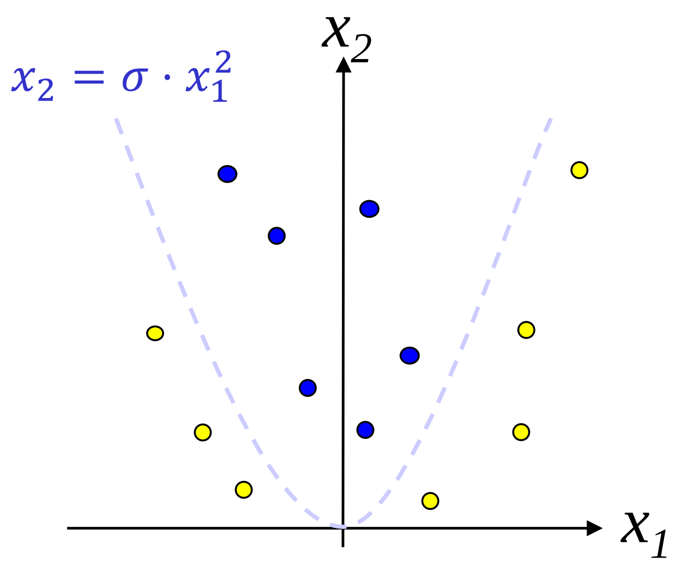
Example

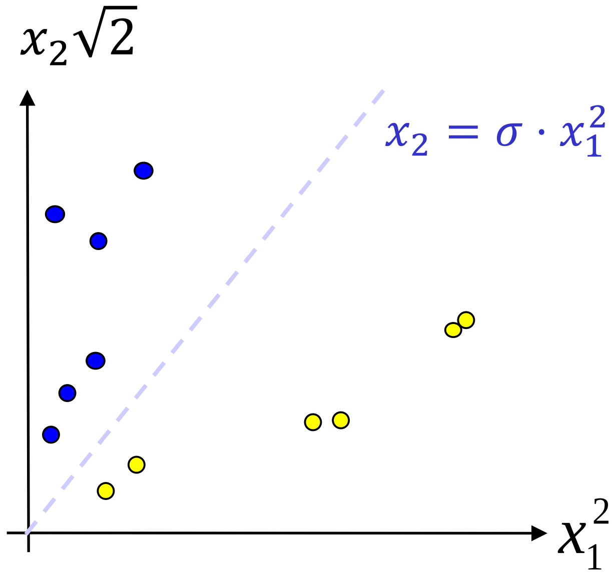
Input space (2 attributes):

$$x = (x_1, x_2)$$

Extended space (6 attributes):

$$\phi(x) = (x_1^2, x_2^2, x_1\sqrt{2}, x_2\sqrt{2}, x_1 x_2\sqrt{2}, 1)$$





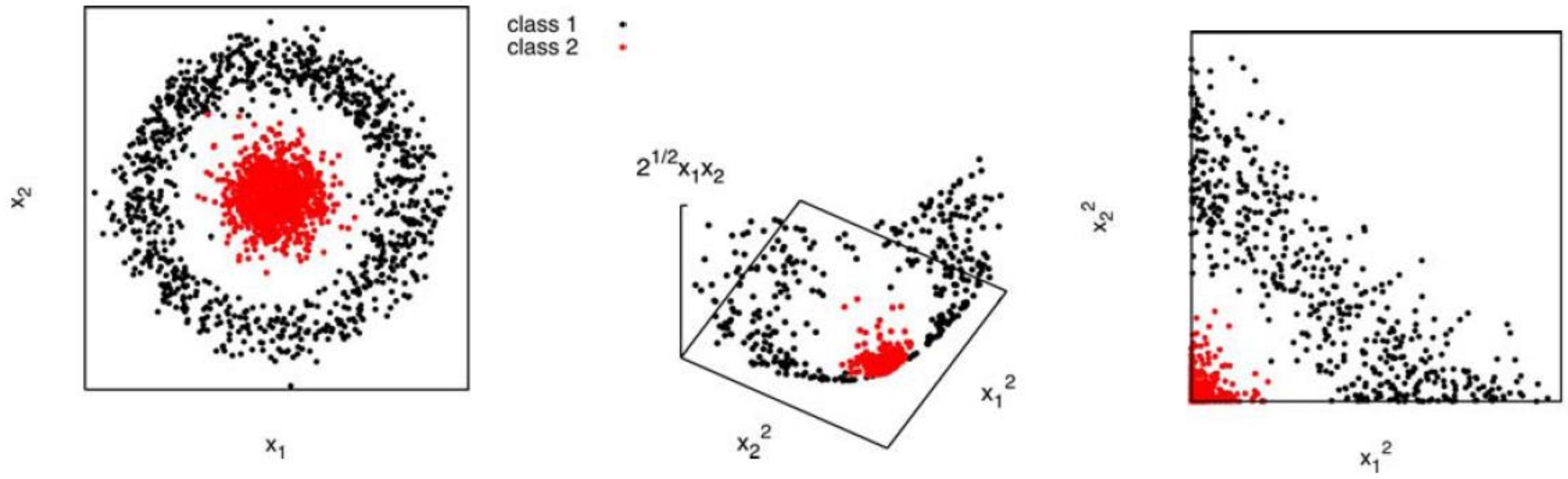
Example (2)

Input space (2 attributes):

$$x = (x_1, x_2)$$

Extended space (3 attributes):

$$\phi(x) = (x_1^2, x_2^2, x_1 x_2 \sqrt{2})$$



DATABASE SYSTEMS GROUP

Extension of linear discriminant function classifier

- Linear classifier can be easily extended to non-linear spaces
- Recap: linear classifier $K_{H(w,w_0)}(x) = sign(w^T x + w_0)$
- Extend to non-linear case
  - Transform all data points o to new feature space $\phi(o)$
  - Data matrix O becomes a matrix Φ
  - The optimal hyperplane vector becomes

Optimal parameter:

$$\tilde{w}_{opt\ \phi} = (\Phi^T \Phi)^{-1} \Phi^{\mathrm{T}} \mathrm{C}$$

  - … and that’s all!
- New classification rule:

Non-linear classification rule:

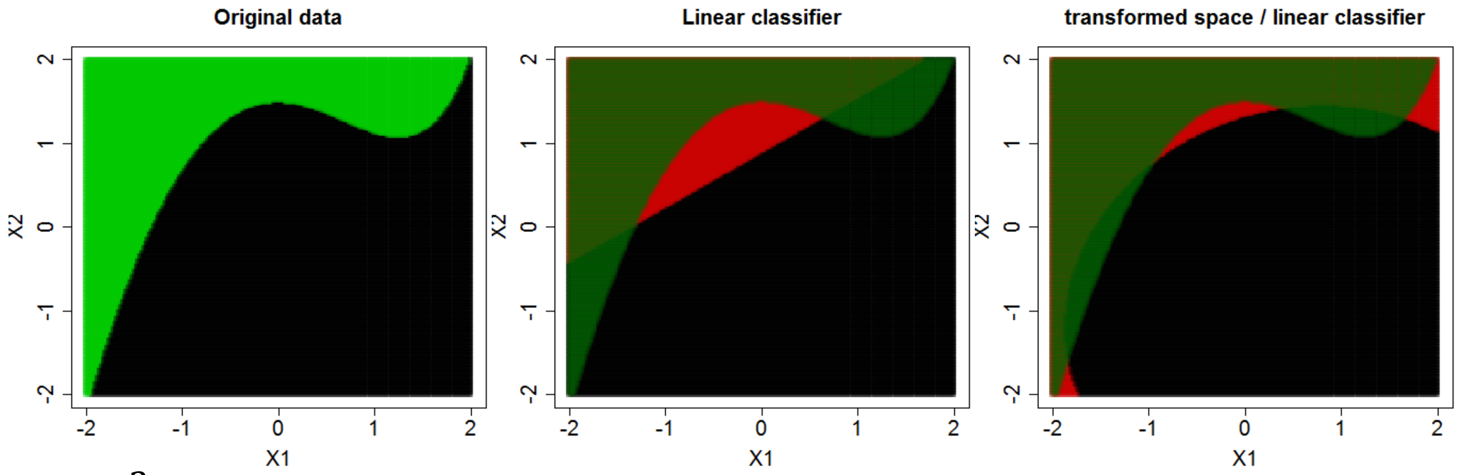
$$K_{H(w,w_0)}(x) = sign\left(w_{opt\ \phi}^T \phi(x) + w_{0\ \phi}\right)$$

- SVM can be extended in a similar way

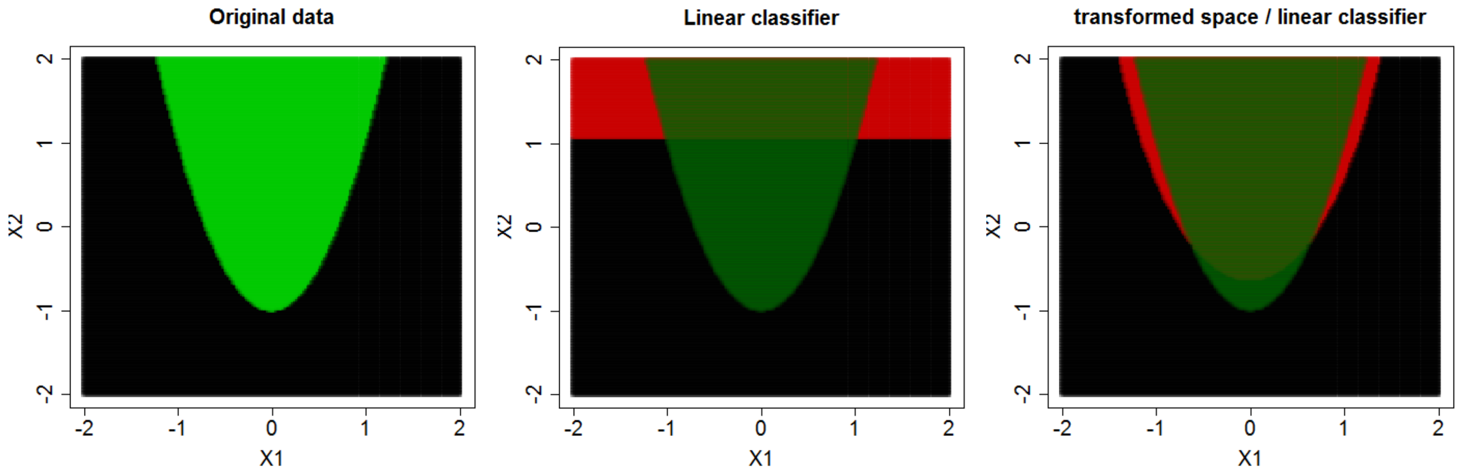
Example

$class\ green: w^T x > \ e^x - 1.3x^2 + 0.1x - 0.5$

$\phi(x) = (x_1^2, x_2^2, x_1\sqrt{2}, x_2\sqrt{2}, x_1 x_2\sqrt{2}, 1)$



$class\ green: w^T x > 2x^2 + 0.3$



Non-linear classification: Discussion

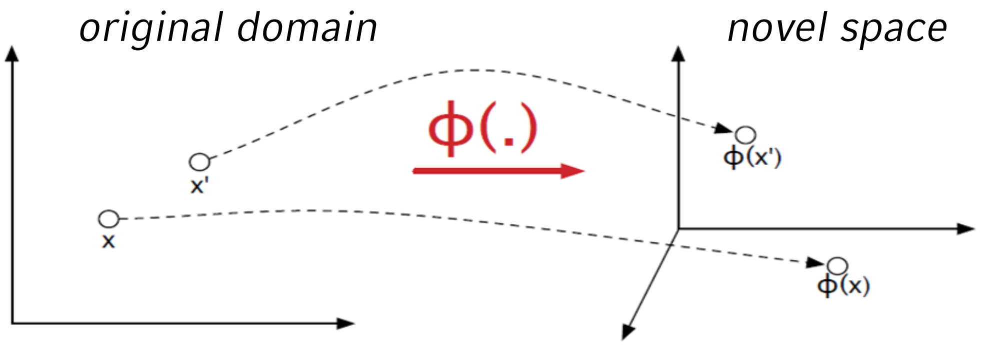
- Pro
  - By explicit feature transformation a much richer hypotheses space
  - Simple extension of existing techniques
  - Efficient evaluation, if transformed feature space not too high-dimensional

- Contra
  - Explicit mapping to other feature spaces can become problematic
  - Meaningful transformation is usually not known a-priori
  - Complex data distributions may require very high-dimensional features spaces
    - High memory consumption
    - High computational costs

- Next: Kernel methods

Implicit Mappings: Kernel Methods

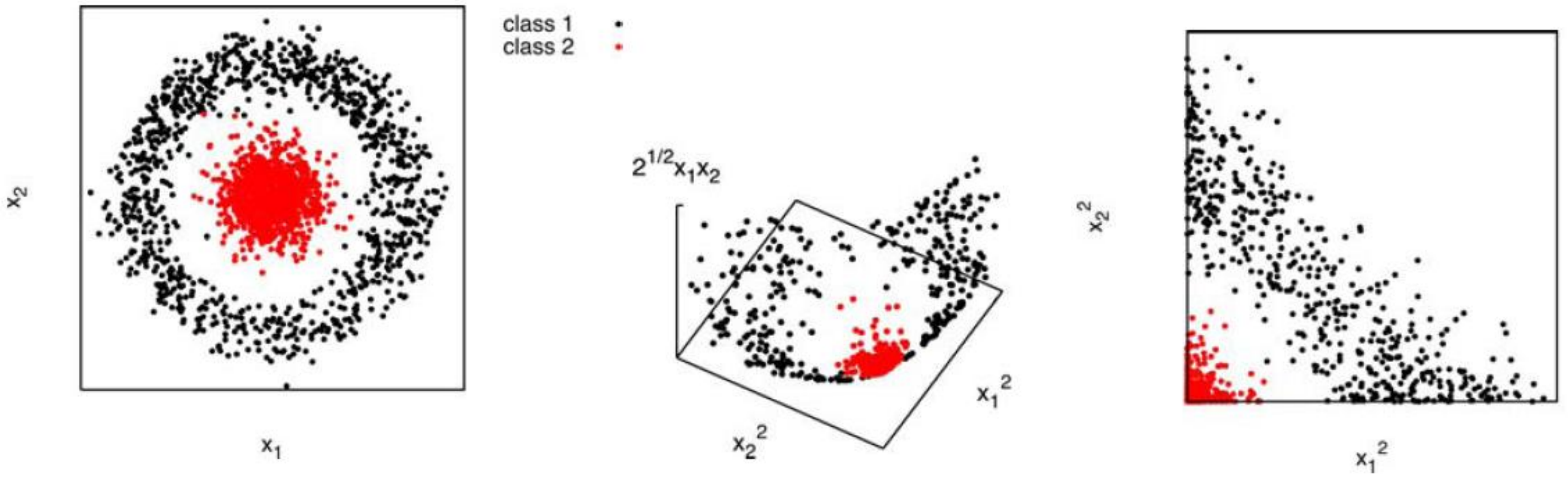
- Explicit mapping of the data into the new feature space:
  - After transformation, any vector-based distance is applied
  - Resulting feature space may be very high dimensional
    - Potential problems: Inefficient calculation, storage overhead
- Often, we do *not* need the transformed data points themselves, but just the distances between them
- Kernel techniques: Just *implicitly* map the data to a feature space
  - Distance calculation between two objects is done in the „original domain“
  - Often called “Kernel trick”



$$K_\phi(x, x') = \langle \phi(x), \phi(x') \rangle$$

Kernel: An example (1)

- Assume our original domain is $\mathcal{X} = \mathbb{R}^2$
- We transform a point $x = (x_1, x_2)$ to $\phi(x) = (x_1^2, x_2^2, \sqrt{2} \cdot x_1 x_2)$
  - i.e. the novel feature space is $\mathcal{H} = \mathbb{R}^3$
  - $\phi: \mathcal{X} \to \mathcal{H}$



[NB07] M. Neuhaus, H. Bunke. *Bridging the Gap Between Graph Edit Distance and Kernel Machines.* 2007.

DATABASE SYSTEMS GROUP

Kernel: An example (2)

LMU

- Original point (x_1, x_2); transformed point $\phi(x) = (x_1^2, x_2^2, \sqrt{2} \cdot x_1 x_2)$
- We want to calculate the dot product in the novel feature space $\mathcal{H}$:

$$\langle x, x' \rangle := \sum_{i=1}^{d} x_i \cdot \mathrm{x}_\mathrm{i}'$$

- What is the dot product between $\phi(x)$ and $\phi(x')$?

$$\begin{aligned}\langle \phi(x), \phi(x') \rangle &= \langle \phi((x_1, x_2)), \phi((x_1', x_2')) \rangle \\ &= x_1^2 \mathrm{x}_1'^2 + x_2^2 \mathrm{x}_1'^2 + 2 x_1 x_2 x_1' x_2' \\ &= (x_1 x_1' + x_2 x_2')^2 \\ &= \langle x, x' \rangle^2\end{aligned}$$

→ We do not have to explicitly map the points to the feature space $\mathcal{H}$!

→ Simply calculate squared dot product in the original domain $\mathcal{X}$!

  – „Kernel Trick“

- $k(x, y) = \langle x, y \rangle^2$ is called a (valid) kernel, $k: \mathcal{X} \rightarrow \mathcal{X}$

Why is the dot product useful?

- Kernels correspond to dot products in some feature space
- With the dot product we are able to compute
  - The norm/length of a vector $\|x\| = \sqrt{\langle x, x\rangle}$
  - The distance between two vectors:
    $$\|x - y\|^2 = \langle x - y, x - y\rangle = \langle x, x\rangle + \langle y, y\rangle - 2\langle x, y\rangle$$
  - The angle between two vectors:
    $$\measuredangle(x, y) = \arccos \frac{\langle x, y\rangle}{\|x\| \cdot \|y\|}$$

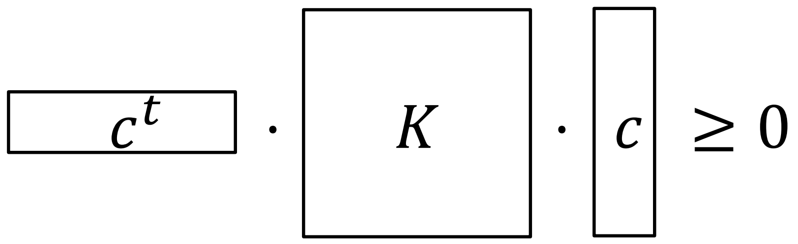
Some Formal Definitions (1)

- Kernel functions
  - A kernel function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric function, $\kappa(x_i, x_j) = \kappa(x_j, x_i)$, mapping pairs of objects $x_i, x_j \in \mathcal{X}$ to real numbers.
- Positive semidefinite kernel functions
  - A kernel function κ is called positive semidefinite if and only if for all $N \in \mathbb{N}$, N constants $\{c_1, .., c_n\} \subseteq \mathbb{R}$ and any choice of N objects $\{x_1, .. x_N\} \subseteq X$ holds:

$$\sum_{i,j=1}^{N} c_i c_j \kappa(x_i, x_j) \geq 0$$

or, in matrix notation, for $c = (c_1, \dots, c_n)^t$ and $K_{ij} = \kappa(x_i, x_j)$:

$$c^t \cdot K \cdot c \geq 0$$



  - The left hand side describes a quadratic form for the matrix K
  - Positive semidefinite kernel functions are often called *valid kernels*, *admissible kernels*, or *Mercer kernels*.

Some Formal Definitions (2)

- **Definition Dot Product :** A dot product in a vector space $\mathcal{H}$ is a function
$$\langle .,. \rangle : \mathcal{H} \times \mathcal{H} \rightarrow \mathbb{R}$$

satisfying:

- $\langle x, x' \rangle = \langle x', x \rangle (Symmetry)$
- $\langle \alpha x + \beta x', x'' \rangle = \alpha \langle x, x'' \rangle + \beta \langle x', x'' \rangle (Bilinearity)$
- $\langle x, x \rangle = 0\ for\ x = 0$
- $\langle x, x \rangle > 0\ for\ x \neq 0$

- **Definition Hilbert Space:** A vector space $\mathcal{H}$ endowed with a dot product $\langle .,. \rangle : \mathcal{H} \times \mathcal{H} \rightarrow \mathbb{R}$ for which the induced norm gives a complete metric space, is termed Hilbert Space

Interpretation of kernel functions

- **Theorem:** Let $\kappa: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ be a valid kernel on a pattern space $\mathcal{X}$.

  There exists a possibly infinite-dimensional Hilbert space $\mathcal{H}$
  and a mapping $\phi: \mathcal{X} \rightarrow \mathcal{H}$ such that
  $\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$ for all $x, x' \in \mathcal{X}$
  where $\langle ., . \rangle$ denotes the dot product in a Hilbert space $\mathcal{H}$

$\rightarrow$ every kernel κ can be seen as a dot product in some feature space $\mathcal{H}$

- Advantages:
  - Feature space $\mathcal{H}$ may be infinite dimensional
  - Not really necessary to know which feature space $\mathcal{H}$ we have
  - Computation of kernel is done in original domain $\mathcal{X}$

Kernel SVM

- Kernel trick can also be used in SVMs:

Seen before, cf. SVM, Soft Margin Optimization

Dual Optimization Problem with Lagrange multipliers (Wolfe dual):

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

HERE: $\sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$

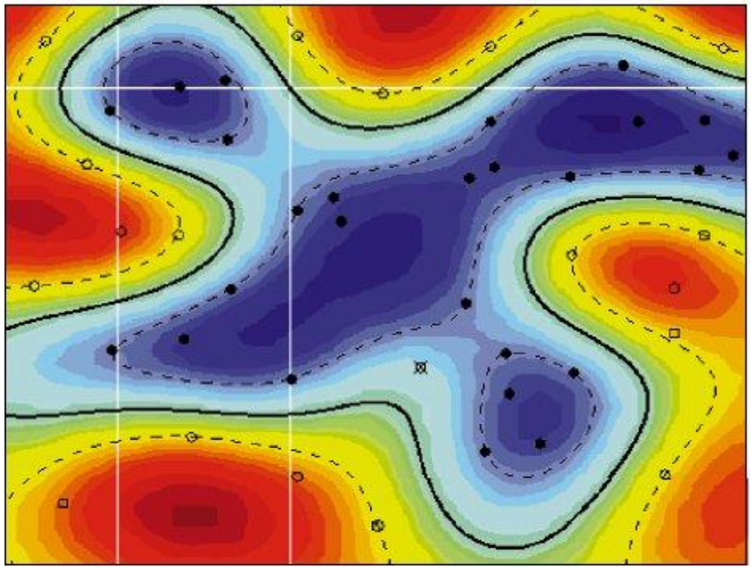
subject to $\sum_{i=1}^{n} \alpha_i \cdot y_i = 0$ and $0 \leq \alpha_i \leq C$

- Feature transform ϕ only affects the scalar product of training vectors
- Kernel K is a function: $K_\phi(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$

Decision rule:

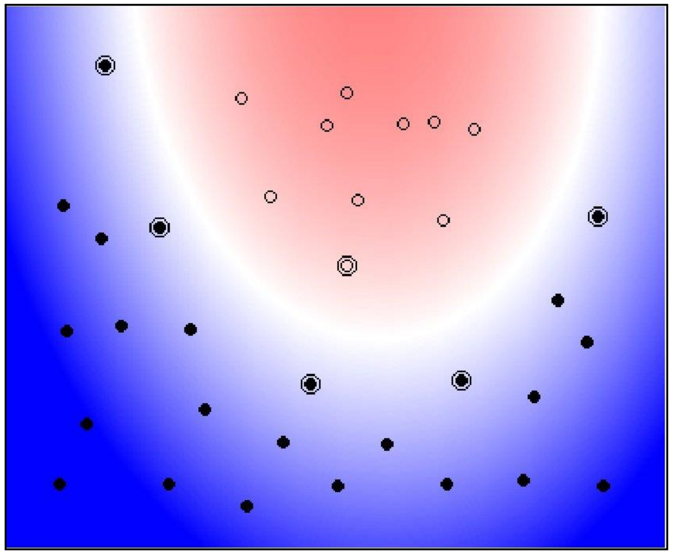
$$h(x) = sign\left(\sum_{x_i \in SV} \alpha_i \cdot y_i \cdot K(\mathbf{x}_i, \mathbf{x}) + b \right)$$

Examples for kernel machines (Mercer kernels)



Radial basis kernel

$$K(\mathbf{x},\mathbf{y}) = \exp\left(-\gamma \cdot |\mathbf{x}-\mathbf{y}|^2\right)$$



Polynomial kernel (degree 2)

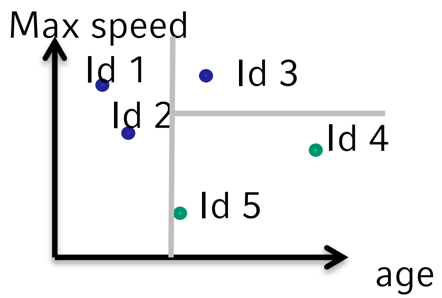
$$K(\mathbf{x},\mathbf{y}) = \left(\langle \mathbf{x},\mathbf{y}\rangle + 1\right)^d$$

Discussion

- Pro
  - Kernel methods provide a simple method for dealing with non-linearity
  - Implicit mapping allows for mapping to arbitrary-dimensional spaces
    - Computational effort depends on the number of training examples, but not on the feature space dimensionality

- Contra
  - resulting models rarely provide an intuition
  - choice of kernel can be difficult

Chapter 6: 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

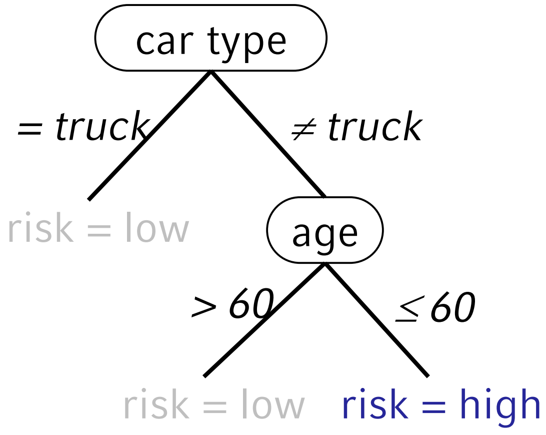
  IF car_type = truck THEN risk = low
  IF car_type ≠ truck AND age > 60 THEN risk = low
  IF car_type ≠ truck AND age ≤ 60 THEN risk = high

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

training data

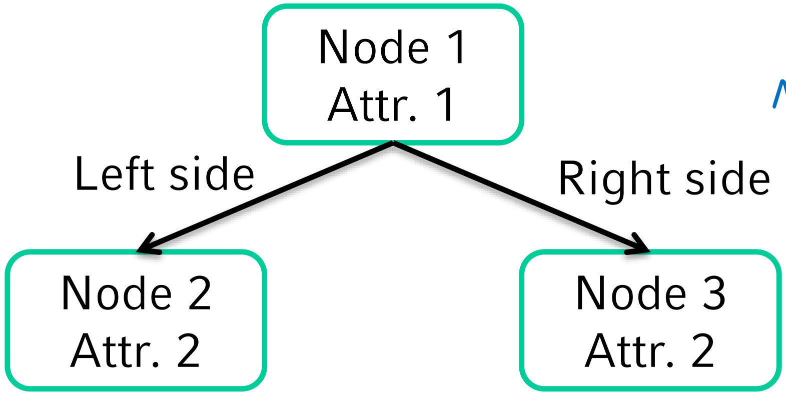
ID	age	car type	risk
1	23	family	high
2	17	sportive	high
3	43	sportive	high
4	68	family	low
5	32	truck	low

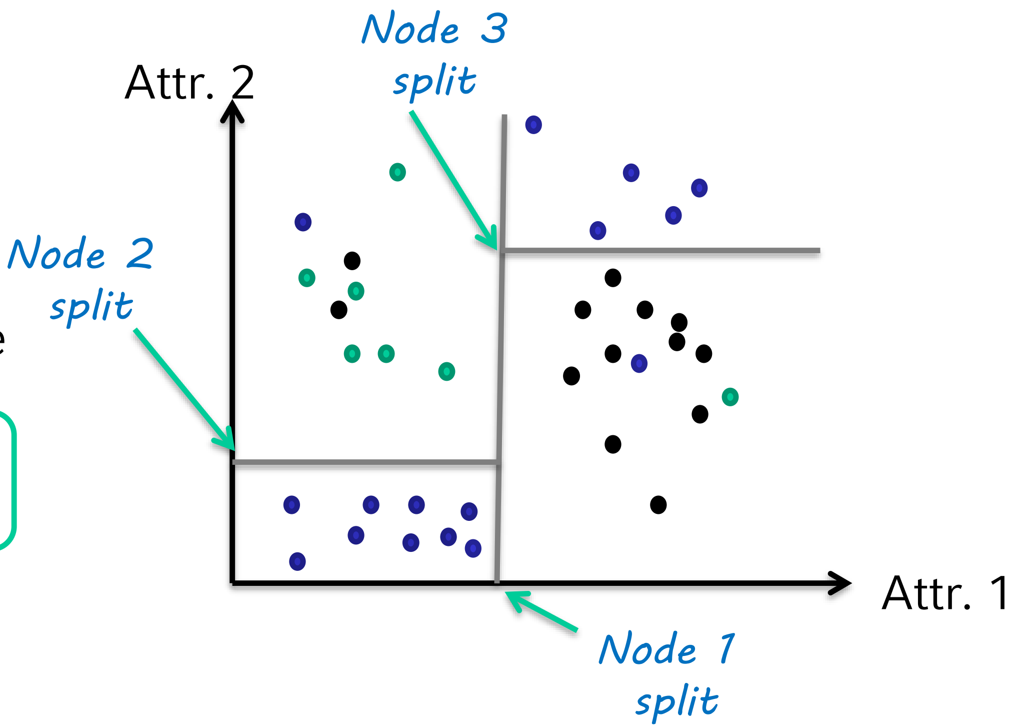
learned decision tree



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=∅, 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“

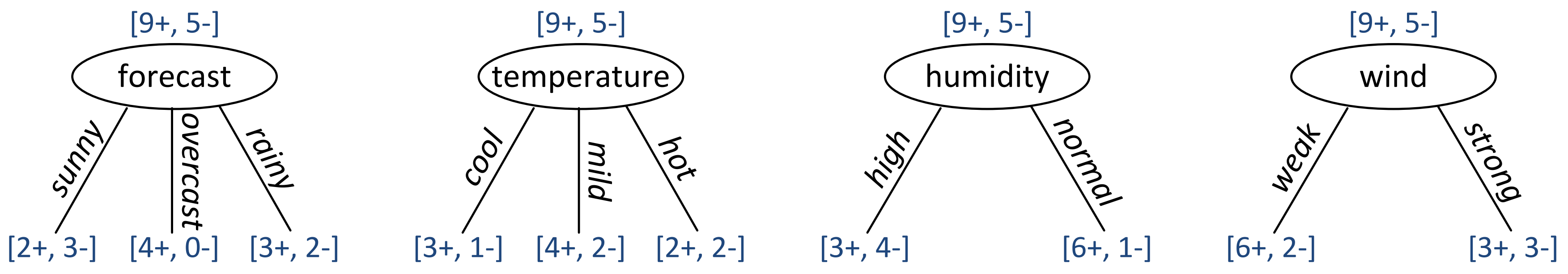
- Query: How about playing tennis today?
- Training data:

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

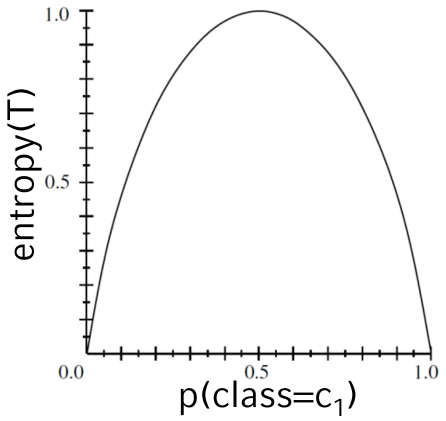
Attribute Selection Measures: Information Gain

- 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:

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

for *k* classes c_i with frequencies p_i

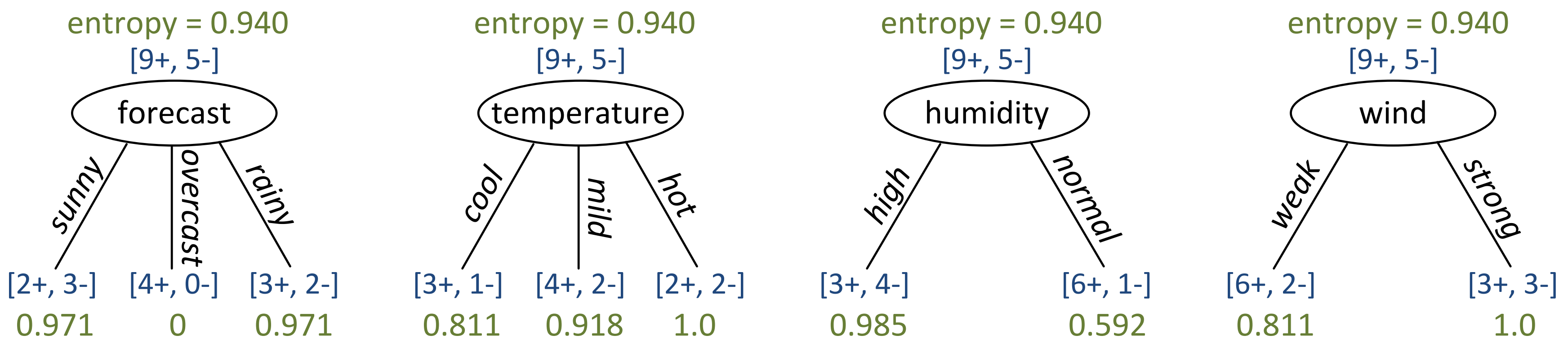
for two classes:



  - 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)



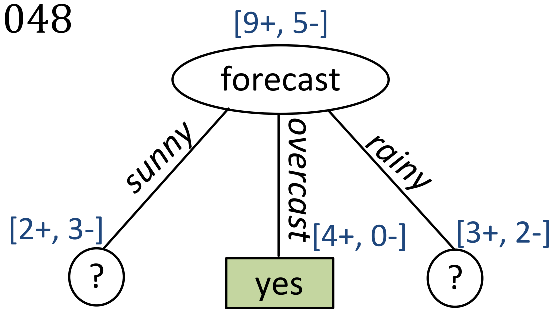
$$information\ gain(T, forecast) = 0.94 - \frac{5}{14} \cdot 0.971 - \frac{4}{14} \cdot 0 - \frac{5}{14} \cdot 0.971 = 0.246$$

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

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

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

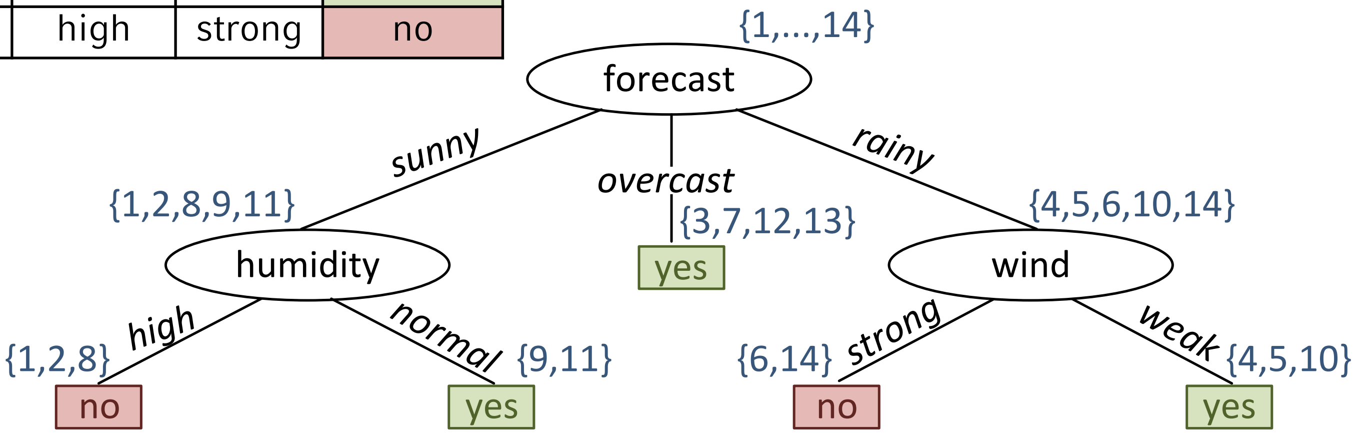
Result: ‘forecast’ yields the highest information gain



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

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

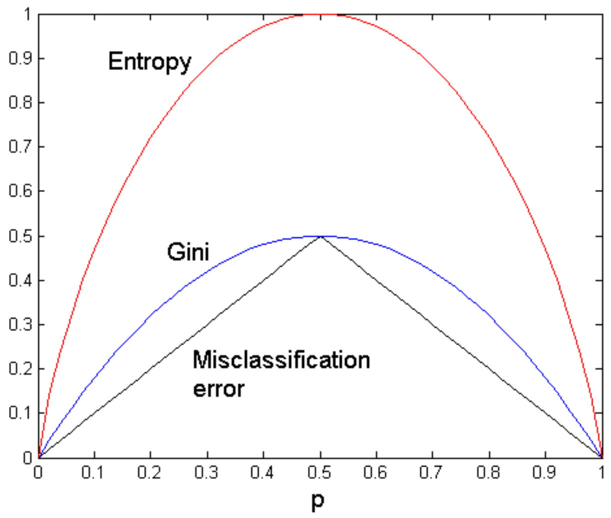
for *k* classes c_i with frequencies p_i

  - small value of Error ⇔ low heterogeneity
  - large value of Error ⇔ high heterogeneity

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

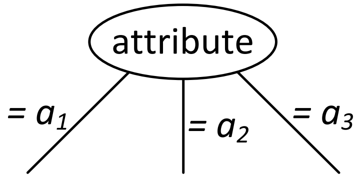
$$Error_A(T) = \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot Error(T_i)$$

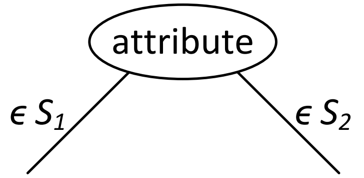
two-class problem:



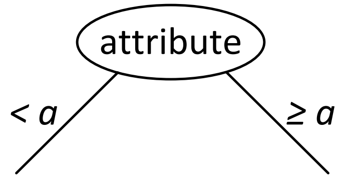
Split Strategies: Types of Splits

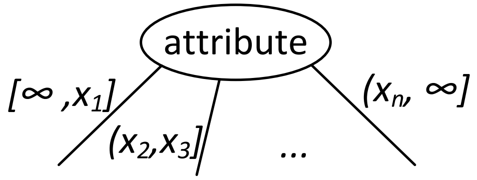
- Categorical attributes
  - split criteria based on equality „*attribute* = *a*"
  - based on subset relationships „*attribute* ϵ *set*"
    → many possible choices (subsets)
    - Choose the best split according to, e.g., gini index





- 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)



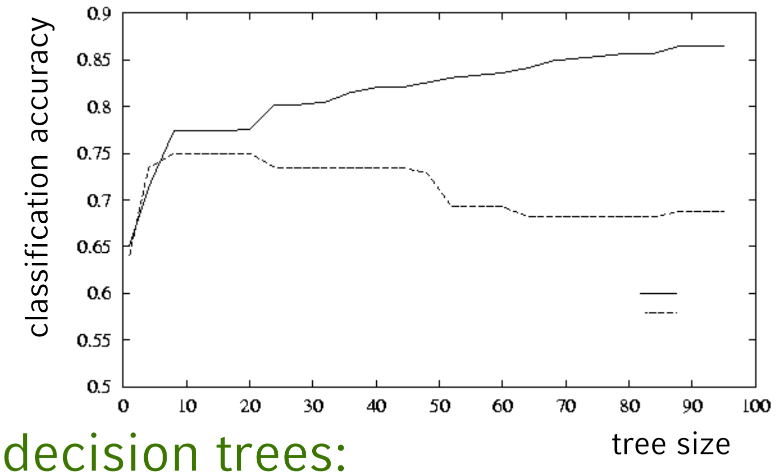


DATABASE SYSTEMS GROUP

Avoid Overfitting in Classification

LMU

- 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.

DATABASE SYSTEMS GROUP

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.

Pruning of Decision Trees: Notions

- Size $|E|$ of a decision tree *E:* number of leaf nodes
- Cost-complexity quality measure of *E* with respect to training set *T* , classification error F_T and complexity parameter $\alpha \geq 0$:

$$CC_T(E,\alpha) = F_T(E) + \alpha \cdot |E|$$

- For the *smallest minimal subtree* $E(\alpha)$ of *E* wrt. α, it is true that:
  - (1) there is no subtree of *E* with a smaller cost complexity
  - (2) if $E(\alpha)$ and *B* both fulfill (1), then is $E(\alpha)$ a subtree of *B*
- $\alpha = 0$: $E(\alpha) = E$ i.e., only error does matter
- $\alpha \to \infty$: $E(\alpha)$ = root node of *E* i.e., only tree size does matter
- $0 < \alpha < \infty$: $E(\alpha)$ is a proper substructure of E, i.e. the root node or more than the root node

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

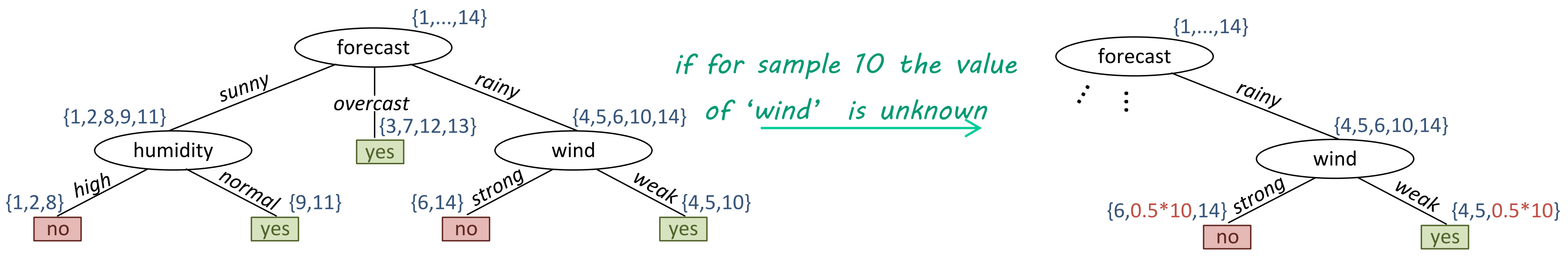
IF forecast = ‘overcast’	THEN playing_tennis = ‘yes’
IF forecast = ‘sunny’ AND humidiy = ‘high’	THEN playing_tennis = ‘no’
IF forecast = ‘sunny’ AND humidiy = ‘normal’	THEN playing_tennis = ‘yes’
IF forecast = ‘rainy’ AND wind = ‘strong’	THEN playing_tennis = ‘no’
IF forecast = ‘rainy’ AND wind = ‘weak’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



    - 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)

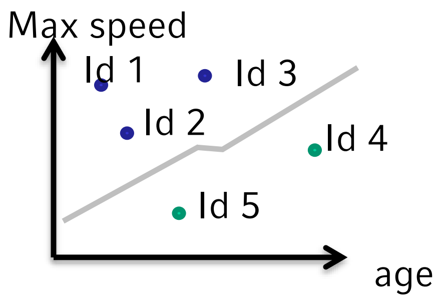
Decision Tree Classifiers: Summary

- 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

Chapter 6: Classification



Nearest Neighbor Classifiers

- Motivation:
  - Assume data in a non-vector representation: graphs, forms, XML-files, etc.
  - No simple way to use linear classifiers or decision trees

- Possible solutions
  - Definition of an appropriate kernel function for kernel machines (e.g. kernel SVM)
    - Not always clear how to define a kernel
  - Transformation of objects to some vector space (multidimensional scaling, histograms for color or shape distributions, etc.)
    - Difficult to choose an appropriate number of dimensions (intrinsic/fractal dimensionality)
  - Here: direct usage of the similarity of objects for classification
    → Nearest neighbor classifier
    - Requires a similarity function

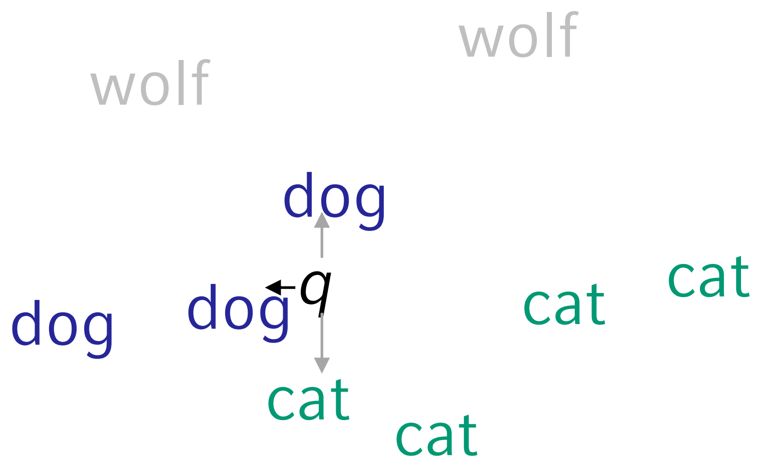
DATABASE SYSTEMS GROUP

Nearest Neighbor Classifiers: Example

- Procedure:

  Assign query object q to the class c_j of the closest training object $o \in O$

$$class(q) = class(NN(q))$$
$$NN(q) = \{o \in O \mid \forall o' \in O: d(o,q) \leq d(o',q)\}$$



- Classifier decides that query object q is a dog
- Instance-based learning

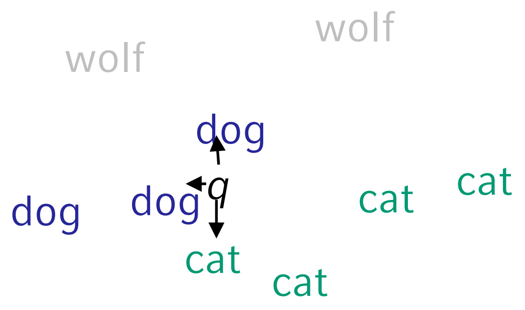
Instance-Based Methods

- Instance-based learning:
  - Store training examples and delay the processing ("lazy evaluation") until a new instance must be classified
  - Typical approaches : *k*-nearest neighbor approach
    - Instances represented as points in an Euclidean space or, more general, as points in a *metric* space
    - Index construction as training phase
      - The classification has to process a huge number of NN-queries → support by e.g. R-tree

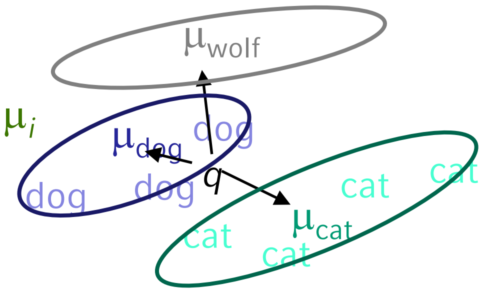
- Eager evaluation
  - Create models from data (training phase) and then use these models for classification (test phase)
  - Examples: Decision tree, Bayes classifier

Nearest Neighbor Classifiers: Variants

- NN Classifier
  - Assign query object to the class c_j of the closest training object
  - Parameter free approach
- *k*-NN Classifier
  - Consider the k>1 nearest neighbors for the class assignment decision
- Weighted *k*-NN Classifier
  - Use weights for the classes of the k nearest neighbors



- Mean-based NN Classifier:
  - Determine mean vector μ_i for each class c_j (in training phase)
  - Assign query object to the class c_j of the nearest mean vector μ_i
- Generalization: representative-based NN Classifier
  - Use more than one representative per class



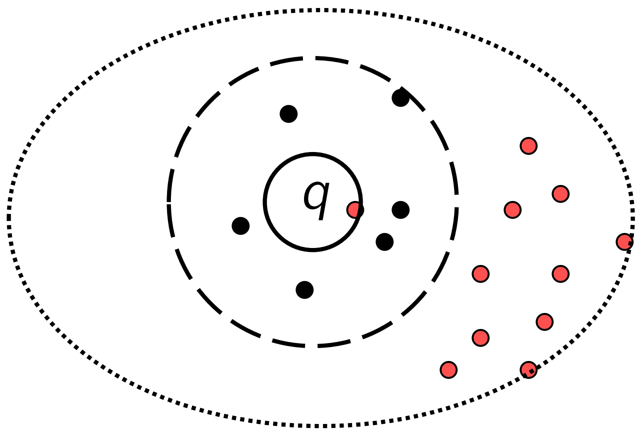
Nearest Neighbor Classifiers: Notions

- Distance function
  - Defines the (dis-)similarity for pairs of objects
- Number *k* of neighbors to be considered
- Decision set
  - Set of *k* nearest neighboring objects to be used in the decision rule
- Decision rule
  - Given the class labels of the objects from the decision set, how to determine the class label to be assigned to the query object?

Nearest Neighbor Classifiers: Parameter *k*

- Tradeoff between *overfitting* and *generalization*:

  Problem of choosing an appropriate value for parameter *k*
  - *k* too small: high sensitivity against outliers
  - *k* too large: decision set contains many objects from other classes
  - Different rules of thumb exist:
    - Based on theoretical considerations: Choose *k*, such that it grows slowly with *n*, e.g. $k \approx \sqrt{n}$ or $k \approx \log(n)$
    - Empirically, 1 << *k* < 10 yields a high classification accuracy in many cases

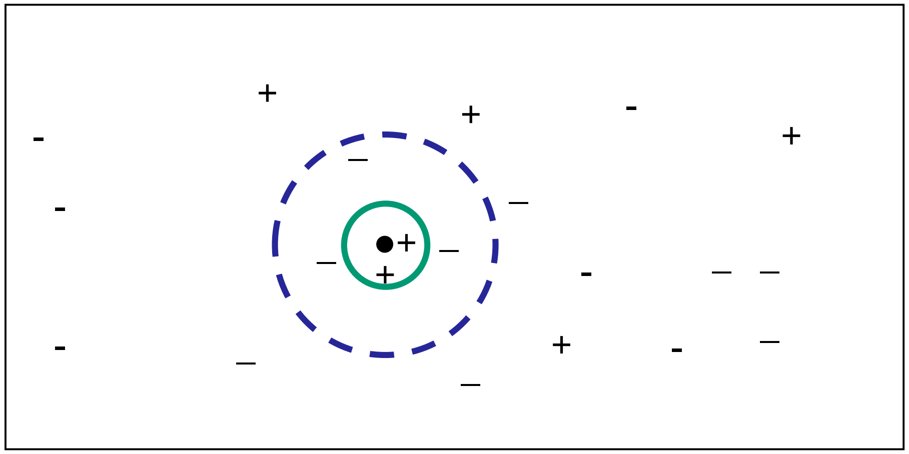


		prediction
◯	decision set for $k = 1$	red
◌	decision set for $k = 7$	black
⋯	decision set for $k = 17$	red

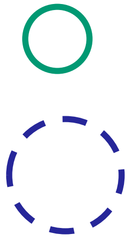
Nearest Neighbor Classifiers: Decision Rules

- Standard rule
  - Choose majority class in the decision set, i.e. the class with the most representatives in the decision set
- Weighted decision rules
  - Use weights for the classes in the decision set
    - Use distance to the query object: $\frac{1}{d(o,q)^2}$
    - Use frequency of classes in the training set, i.e. the *a-priori probability* of the class
  - Example
    - Class a: 95%, class b: 5%
    - Decision set = {**a**, **a**, **a**, **a**, **b**, **b**, **b**}
    - Standard rule yields class "**a**"
    - Weighted rule yields class "**b**" (a-priori based)

Nearest Neighbor Classifiers: Example



Classes + and –



◯ decision set for $k = 2$

◌ decision set for $k = 5$

- Using unit weights (i.e., no weights) for the decision set
  - Simply called *"majority criterion"*
  - rule $k = 2$ yields class „+", rule $k = 5$ yields class „–"
- Using the reciprocal square of the distances as weights
  - Both rules, $k = 2$ and $k = 5$, yield class „+"
- Using a-priori probability (=frequency) of classes as weights
  - Both rules, $k = 2$ and $k = 5$, yield class „+"

Nearest Neighbor Classification: Discussion

- Pro
  - applicability: training data and distance function required only
  - high classification accuracy in many applications
  - easy incremental adaptation to new training objects
  - useful also for prediction
  - robust to noisy data by averaging *k*-nearest neighbors
- Contra
  - naïve implementation is inefficient
    - requires k-nearest neighbor query processing
    - support by database techniques may help to reduce from $O(n)$ to $O(\log n)$ for n training objects → training phase: create index structure
  - does not produce explicit knowledge about classes
    - But provides some explanation information
  - Curse of dimensionality: distance between neighbors could be dominated by irrelevant attributes
    - To overcome it, stretch axes or eliminate least relevant attributes

Chapter 5: Classification

Ensemble Classification

- No single classifier performs good on every problem
- For some techniques, small changes in the training set lead to very different classifiers

- Idea: improve performance by *combining* different classifiers

  => ensemble classification

- Different possibilities exist
- Discussed here:
  - Bagging (Bootstrap aggregation)
  - Boosting

Bagging

- How do we get different classifiers?
  - Easiest way: Train the *same classifier* K on *different datasets*

- Bagging (or *Bootstrap Aggregation*):
  - Randomly select M different subsets from the training set
  - On each subset, train a classifier K_M

  - Overall decision: $K(o) = sign(\frac{1}{M}\sum_1^M K_M(o))$

Boosting

- A technique for combining multiple 'base classifiers'
- Can produce good results even if the single classifiers are only slightly better than random guessing!

- Linear combination of several *weak learners (different classifiers)*
- Given M weak learners $\{K_1, \dots, K_M\}$ and M weights $\alpha_1, \dots, \alpha_M$(with $\sum_{i=1}^{M} \alpha_i = 1$)
- The final classifier K is given by $\mathrm{K}(x) = sign\left(\sum_{i=1}^{M} \alpha_i \, K_i(x)\right)$

- Difference to bagging:
  - Here, the classifiers are trained in sequence
  - Each classifier is trained to perform better on data points that were misclassified by previous classifiers

DATABASE SYSTEMS GROUP

AdaBoost

- Widely used boosting method: AdaBoost (Adaptive Boosting) [FS1996]
  - Meta-algorithm that iteratively generates a chain of weak learners
- General idea: Assume (t-1) weak learners are already given. The t^{th} learner should focus on instances that were previously misclassified.
- Assign a weight w_i to each instance x_i to represent its importance
- Start with equal weight for each instance, adapt weights according to the performance of previously trained classifiers

[FS1996] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1):119–139, 1996.

AdaBoost - Algorithm

- Initialize $w_1, \dots, w_N = \frac{1}{N}$ $\quad (N\text{: } number\ of\ training\ instances)$
- For $m = 1, \dots, M$:
  - Fit a classifier $K_m(x)$ to the training data by minimizing weighted error function $J_m = \sum_{n=1}^{N} w_n I(K_m(x_n) \neq t_n)$ $\quad$ (t_n: correct class label, I: indicator function)
  - Compute weighting coefficient $\alpha_m = \ln\left\{\frac{1-\epsilon_m}{\epsilon_m}\right\}$ with $\epsilon_m = \frac{J_m}{\sum_1^N w_n}$
  - Updata all data weights: $w_{n,old} = w_n$
    $w_n = w_{n,old} \exp\{\alpha_m I(K_m(x_n) \neq t_n)\}$
- Make prediction using final model

$$K_M(x) = sign\left(\sum_{m=1}^{M} \alpha_m K_m(x)\right)$$

Classification: Summary

	Linear models	SVM	Decision Trees	k-NN classifier	Bayes classifier
Compactness	Usually compact (number dims)	Compact if few #supp. vectors	Compact if pruned	No model	Model dependent
Interpretability of model	Medium-Low	Medium-Low	Good	-	Model dependent
Explanation of decision	Low	Low	Good rules for decision known	Medium-Good decision object set known	Medium-Good probabilities of decision are given
Training time	High	Medium	Low-Medium	No training	Model dependent
Test time	Low-High (if high dimensional)	Low-Medium	Low	Low (index) Very high	Model dependent but often Low
Robustness	Low	High	Low	High	High
Data types	Arbitrary data (kernel-dependent)	Arbitrary data (kernel-dependent)	Categorical and vector	Arbitrary data (need distance function)	Arbitrary data (need probability distribution)
Model	Hyperplane	Hyperplane or non-linear (kernel)	Set of (axis parallel) hyperplanes	Model free	Statistical density distribution

Discussion: Choice of classifiers

- There are different classifiers…
  - are there any reasons to prefer one classifier over another if we make no assumptions about the nature of the classification task?
  - Or is there maybe even an overall superior algorithm to random guessing?

- The answer to these and many related questions is: NO

- Also known as "No Free Lunch Theorem"

Classification Chapter – Conclusions

- Classification is an extensively studied problem (mainly in statistics and machine learning)
- Classification is probably one of the most widely used data mining techniques with a lot of extensions
- Scalability is an important issue for database applications: thus combining classification with database techniques should be a promising topic
- Research directions: classification of complex data, e.g., text, spatial, multimedia, etc.
  - Example: kNN-classifiers rely on distances but do not require vector representations of data
- Results can be improved by ensemble classification