Ludwig-Maximilians-Universität München
Lehrstuhl für Datenbanksysteme und Data Mining
Prof. Dr. Thomas Seidl

Knowledge Discovery and Data Mining 1

(Data Mining Algorithms 1)

Winter Semester 2019/20

Agenda

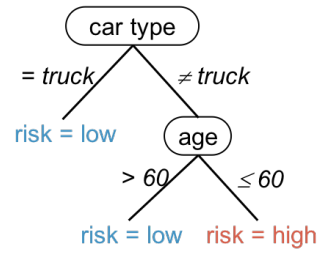
Decision Tree Classifiers

- 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
- Tree can be transformed into decision rules:
  if age > 60 **then** risk $=$ low
  if age ≤ 60 **and** car type $=$ truck **then** risk $=$ low
  if age ≤ 60 **and** car type $\neq$ truck **then** risk $=$ high

Advantages

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

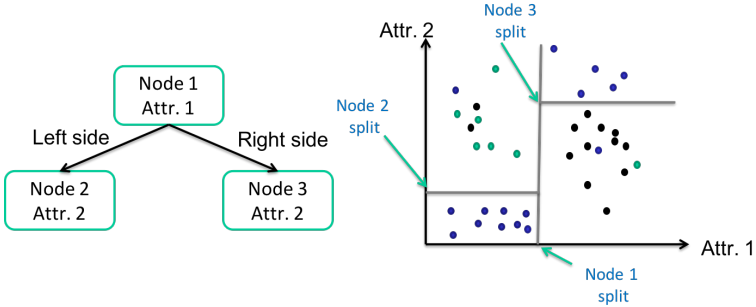
age	car_type	max_speed	risk
23	family	180	high
17	sportive	240	high
43	sportive	246	high
68	family	183	low
32	truck	110	low



Decision Tree Classifier: Splits

Goal

- Each tree node defines an axis-parallel $(d-1)$-dimensional hyperplane, that splits the data space.
- *Find such splits which lead to as homogeneous 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 the start, all the training examples are assigned to the root node
  - Recursively partition 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 based on 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, or its successor C4.5

ID3 Algorithm

procedure ID3(*Examples*, *TargetAttr*, *Attributes*) ▷ specialized to learn boolean-valued functions
  Create *Root* node for the tree
  if all *Examples* are positive **then return** *Root* with *label* $= +$
  else if all *Examples* are negative **then return** *Root* with *label* $= -$
  else if *Attributes* $= \emptyset$ **then return** *Root* with *label* $=$ most common value of *TargetAttr* in *Examples*
  else
    $A =$ "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 **do** ▷ 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 $Examples_{v_i} = \emptyset$ **then**
        Add leaf node with *label* $=$ most common value of *TargetAttr* in *Examples*
      else
        Add subtree ID3($Examples_{v_i}$, *TargetAttr*, *Attributes* $\setminus \{A\}$)

Example: Decision for "playing_tennis"

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

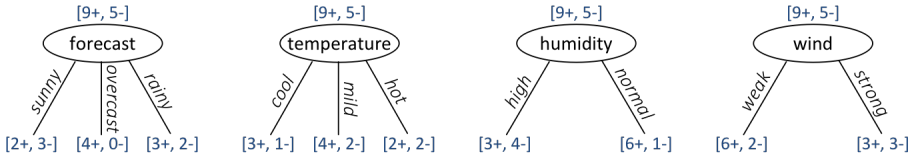
day	forecast	temperature	humidity	wind	tennis decision
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rainy	mild	high	weak	yes
5	rainy	cool	normal	weak	yes
6	rainy	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	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 $T_1, \ldots T_m$ of T
- The relative frequencies p_i of class c_i in T and in the partitions $T_1, \ldots T_m$



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 $\{T_1, \ldots, T_m\}$ such that the heterogeneity is minimized

$\rightsquigarrow$ Proposals: *Information gain*, *Gini index*, *Misclassification error*

Attribute Selection Measures: Information Gain

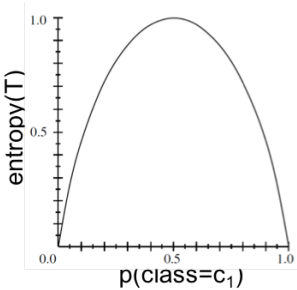
- Used in ID3/C4.5

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

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

  for k classes with frequencies p_i
- $entropy(T) = 0$ if $p_i = 1$ for any class c_i
- $entropy(T) = 1$ if $p_i = \frac{1}{k}$ for all classes c_i



$k = 2$

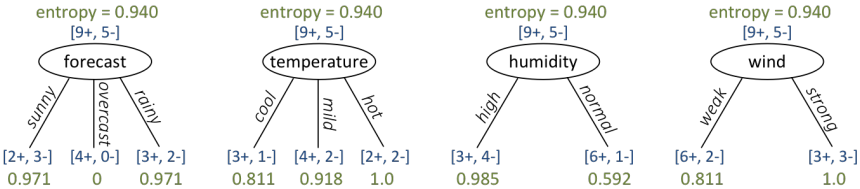
Attribute Selection Measures: Information Gain

Information Gain

Let A be the attribute that induced the partitioning $\{T_1, \ldots, T_m\}$ of T. The information gain of attribute A w.r.t. T is defined as

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

Attribute Selection: Example (Information Gain)

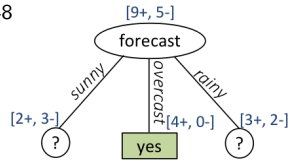


$$information\_gain(T, forecast) = 0.94 - \frac{5}{14}0.971 - \frac{4}{14}0 - \frac{5}{14}0.971 = 0.246$$

$$information\_gain(T, temperature) = 0.94 - \frac{4}{14}0.811 - \frac{6}{14}0.981 - \frac{4}{14}1 = 0.029$$

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

$$information\_gain(T, wind) = 0.94 - \frac{8}{14}0.811 - \frac{6}{14}1 = 0.048$$

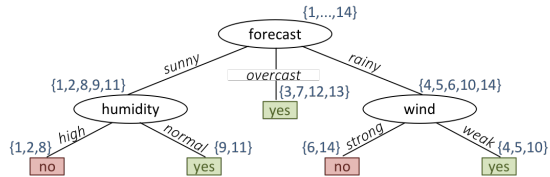


Result: "forecast" yields the highest information gain

Example: Decision Tree for "playing_tennis"

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

Final decision tree:



Attribute Selection Measures: Gini Index

- Used in IBM's IntelligentMiner

Gini Index

The Gini index for a set T of training objects is defined as

$$gini(T) = 1 - \sum_{i=1}^{k} p_i^2$$

- Small value of Gini index $\equiv$ low heterogeneity
- Large value of Gini index $\equiv$ high heterogeneity

Gini Index (of an attribute A)

Let A be the attribute that induced the partitioning $\{T_1, \ldots, T_m\}$ of T. The Gini index of attribute A w.r.t. T is defined as

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

Attribute Selection Measures: Misclassification Error

Misclassification Error

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

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

- Small value of Error $\equiv$ low heterogeneity
- Large value of Error $\equiv$ high heterogeneity

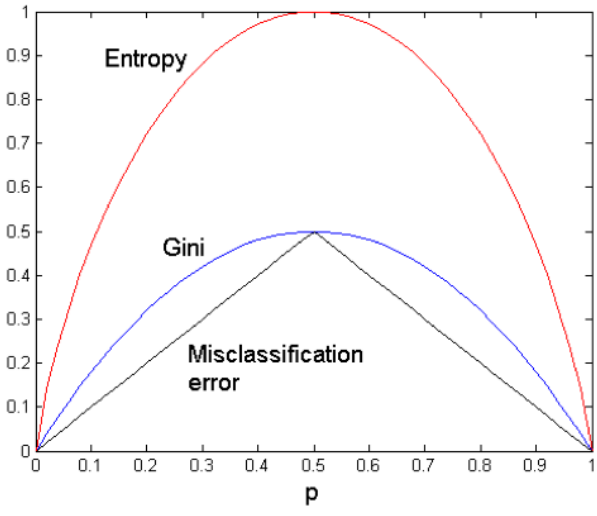
Misclassification Error (of an attribute A)

Let A be the attribute that induced the partitioning $\{T_1, \ldots, T_m\}$ of T. The Misclassification Error of attribute A w.r.t. T is defined as

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

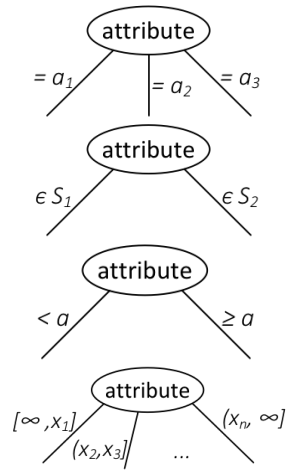
Attribute Selection Measures: Comparison

For two-class problems:



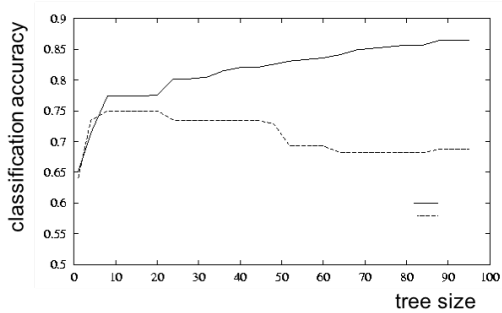
Split Strategies: Types of Splits

- Categorical attributes
  - Split criteria based on equality "$attribute = a$"
  - Based on subset relationships "$attribute \in set$"
    $\rightsquigarrow$ many possible choices (subsets)
    - Choose the best split according to, e.g., gini index
- Numerical attributes
  - Split criteria of the form "$attribute < a$"
    $\rightsquigarrow$ 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
  2. Pre-pruning = halt tree construction early

Pruning Techniques for Decision Trees

Post-pruning

Pruning of overspecialized branches:

- Remove branches from a "fully grown" tree and 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"

Pruning Techniques for Decision Trees

Pre-pruning

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* $\gg 1$
- Choice of an appropriate value for *minimum confidence*
  - Minimum confidence: minimum fraction of the majority class in a leaf node
  - Typically, *minimum confidence* $\ll 100\%$
  - Leaf nodes can absorb errors or noise in data records
- Discussion
  - With Pre-pruning it is difficult to choose appropriate thresholds
  - Pre-pruning has less information for the pruning decision than post-pruning $\rightsquigarrow$ can be expected to produce decision trees with lower classification quality
  - Tradeoff: tree construction time vs. classification quality

Avoid Overfitting with Regularization

General: Regularization

Solve the regularized minimization problem

$$\min_{\theta} f(\cdot, \theta) + \lambda g(\theta)$$

where θ denotes the model's parameters, $f(\cdot, \theta)$ is used as a loss function, $g(\theta)$ is a **regularization term** and λ is a trade-off hyperparameter.

- Regularization terms are used to fine-tune the model's complexity
  - Prevents overfitting of a model
- The L_1-norm and L_2-norm, respectively, are commonly used for regularizing the model's parameter

Minimal Cost Complexity Pruning: 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 $\alpha \geq 0$:

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

- For the *smallest minimal subtree* $E(\alpha)$ of E w.r.t. 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
- $\alpha = 0$: $E(\alpha) = E$
  - Only error matters
- $\alpha \to \infty$: $E(\alpha) =$ root node of E
  - Only tree size matters
- $0 < \alpha < \infty$: $E(\alpha)$ is a proper substructure of E
  - The root node or more than the root node

Decision Tree Classifiers: Summary

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

Agenda

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

Solutions

- Use appropriate kernel function for kernel machines (e.g. kernel SVM)
  $\rightsquigarrow$ Not always clear how to define a kernel
- Embedding of objects into some vector space (e.g. representation learning)
  $\rightsquigarrow$ Difficult to determine appropriate embedding projection
- *Here*: Nearest neighbor classifier
  $\rightsquigarrow$ Direct usage of (dis-)similarity functions for objects

Nearest Neighbor Classifiers

Procedure

Assign query object q to the class c_j of the closest training object $x \in D$:

$$class(q) = class(NN(q)) \qquad NN(q) = \{x \in D \mid \forall x' \in D : d(q,x) \leq d(q,x')\}$$

Example

Classifier decides that query object q is a dog.

Instance-Based Learning

Eager Evaluation

- Examples: Decision tree, Bayes classifier, SVM
- Training phase: Learn parameters for chosen model from training data
- Test phase: evaluate parameterized model for arriving query objects

Lazy Evaluation

- Typical Approach: (k-)nearest neighbor classifiers
- Derive labels from individual training objects: *instance-based learning*
- No training required (= lazy)
- Highly recommended: put data into efficient index structure (e.g., R-Tree)

Nearest Neighbor Classifiers: Notions

Notions

- Distance Function: Defines the (dis-)similarity for pairs of objects
- *Decision Set*: The set of k nearest neighboring objects used in the decision rule

Decision Rule

Given the class labels of the objects from the decision set, how to derive the class label for the query object?

(Plain) Decision Rules

Given a query instance x_q and its k nearest neighboring training objects, $(x_i)_{i=1}^k$. Let $\delta(\cdot,\cdot)$ denote the *Kronecker delta* and $C_i = C(x_i)$ be the class label of x_i:

Nearest Neighbor Rule ($k = 1$)

Just inherit the class label of the nearest training object:

$$K(x_q) = C(x_1)$$

Majority Vote ($k \geq 1$)

Choose majority class, i.e. the class with the most representatives in the decision set:

$$K(x_q) = \underset{c_j \in \mathcal{C}}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(C_i, c_j)$$

Weighted Decision Rules

Distance-weighted majority vote

Give more emphasis to closer objects within decision set, e.g.:

$$K(x_q) = \underset{c_j \in \mathcal{C}}{\operatorname{argmax}} \sum_{i=1}^{k} \frac{\delta(C_i, c_j)}{dist(x_i, x_q)^2}$$

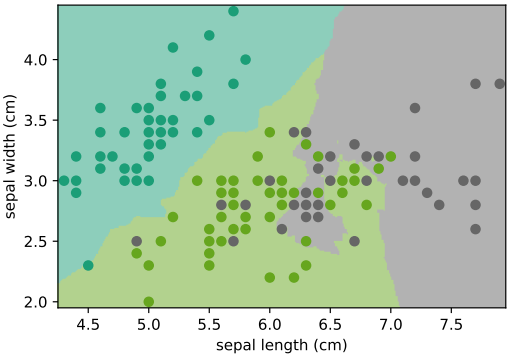
Class-weighted majority vote

Use inverse frequency of classes in the training set (a-priori probabilities):

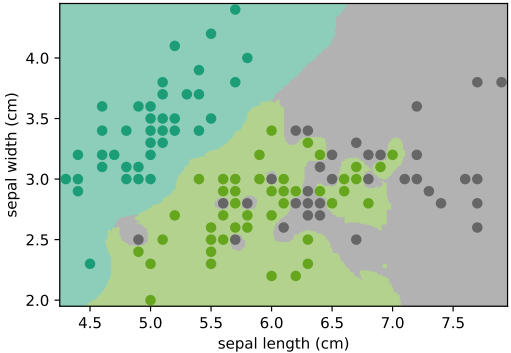
$$K(x_q) = \underset{c_j \in \mathcal{C}}{\operatorname{argmax}} \frac{\sum_{i=1}^{k} \delta(C_i, c_j)}{\sum_{x \in O_{TR}} \delta(C(x), c_j)}$$

Example: Influence of Weighting (here: $k = 5$)

(Plain) Majority Vote



Reciprocal Squared Distance



NN Classifier: Parameter k

Choosing an appropriate k: Tradeoff between *overfitting* and *generalization*:

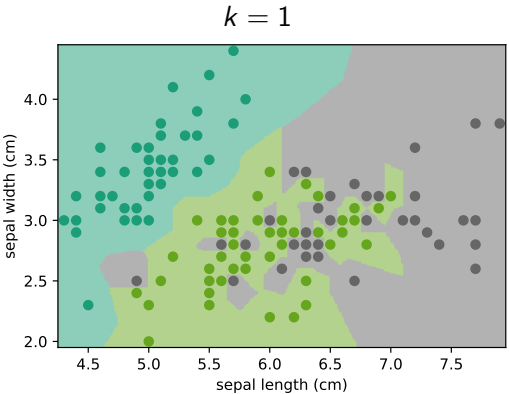
Influence of k

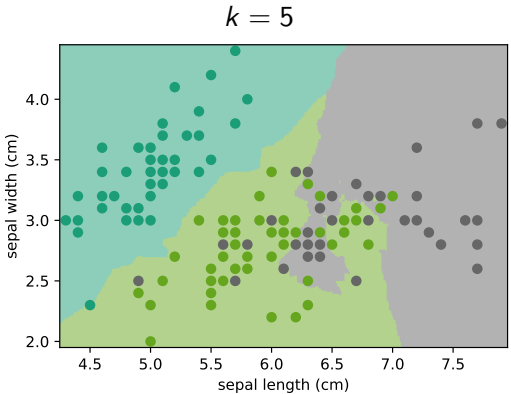
- k too small: High sensitivity against outliers
- k too large: Decision set contains many objects from other classes

Rules of Thumb

- 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, $1 \ll k < 10$ yields a high classification accuracy in many cases

Example: Majority Vote – Influence of k





NN Classifier: Variants

- *k-NN* Classifier: Consider the k 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 m_i for each class c_j (in training phase); Assign query object to the class c_j of the nearest mean vector m_i
- *Generalization*: Representative-based NN mean classifier; use more than one representative per class (cf. mixture models) – no longer just instance-based

NN Classifier: 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 $\rightsquigarrow$ support by database techniques may help to reduce from $\mathcal{O}(n)$ to $\mathcal{O}(\log n)$
- Does not produce explicit knowledge about classes, but provides explanations
- Curse of dimensionality: Distance between neighbors could be dominated by irrelevant attributes $\rightsquigarrow$ apply dimensionality reduction first

Agenda

Ensemble Classification

Problem

- No single classifier performs good on every problem
  (cf. theorem "There is no free lunch")
- For some techniques, small changes in the training set lead to very different classifiers

Idea

Improve performance by combining different classifiers $\rightsquigarrow$ ensemble classification.
Different possibilities exist. Discussed here:

- Bagging (Bootstrap aggregation)
- Boosting

Bagging

How to obtain 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, independently train a classifier K_i $(i = 1, \ldots, m)$
- Overall decision:

$$K(x) = sign\left(\frac{1}{m}\sum_{i=1}^{m} K_i(x)\right)$$

Boosting

- Linear combination of several *weak* learners (different classifiers)
- Given m weak learners K_i and weights α_i for $i = 1, \ldots, m$
- Overall decision

$$K(x) = sign\left(\sum_{i=1}^{m} \alpha_i K_i(x)\right)$$

- *Important difference*: classifiers are trained in sequence!
- Repeatedly misclassified points are weighted stronger
- Example: meta-algorithm *AdaBoost* iteratively generates a chain of weak learners

Classification: Summary

	Linear Model	SVM	Decision Tree	kNN	Bayes
Model	hyperplane	hyperplane/ non-linear (kernel)	hierarchy of iso-oriented hyperplanes	no model	probability distribution func's
Data Types	vectors/kernels	vectors/kernels	categorical & vector	metric, kernels	arbitrary
Compactness	good (#dims)	good (#SV)	good (pruned)	no model	depends/model
Interpretability of Model	medium/low	medium/low	good	no model	depends/model
Interpretability of Decision	low	low	good (rules)	medium/good (examples)	medium/good (probabilities)
Training Time	high	medium	low/medium	no training	depends/model
Test Time	low / high (if high-dim)	low/medium	low	low w/ index, high w/o index	depends/model, often low
Robustness	low	high	low	high	high

Classification: Conclusion

- 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