

## 7. Classification

### Knowledge Discovery in Databases

Dominik Probst, [Dominik.probst@fau.de](mailto:Dominik.probst@fau.de)

Chair of Computer Science 6 (Data Management), Friedrich-Alexander-University Erlangen-Nürnberg

Summer semester 2024

---

# Outline

1. **Basic Concepts**
2. **Decision Tree Induction**
3. **Rule-Based Classification**
4. **Bayes Classification Methods**
5. **Model Evaluation and Selection**
6. **Ensemble Methods: Increasing the Accuracy**
7. **Summary**
8. **Appendix**

---

# Basic Concepts

# Supervised vs. Unsupervised Learning

## Supervised Learning

- The **training data** (observations, measurements, etc.) are accompanied by **labels** indicating the **class** of the observations.
- New data is classified based on a **model** created from the training data.

## Unsupervised Learning

- Class labels of training data are unknown. Or rather, there are no training data.
- Given a set of measurements, observations, etc., the goal is to find classes or clusters in the data.  
→ See next chapter (Lecture/Chapter 8: Clustering).

# Prediction Problems: Classification vs. Numerical Prediction

- **Classification:**

- Predicts **categorical class labels** (discrete, nominal).
- Constructs a model based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data.

- **Numerical prediction:**

- Models **continuous-valued functions**.
- I.e. predicts missing or unknown (future) values.

- **Typical applications of classification:**

- Credit/loan approval: Will it be paid back?
- Medical diagnosis: Is a tumor cancerous or benign?
- Fraud detection: Is a transaction fraudulent or not?
- Web-page categorization: Which category is it such as to categorize it by topic.

# Classification – A Two-step Process

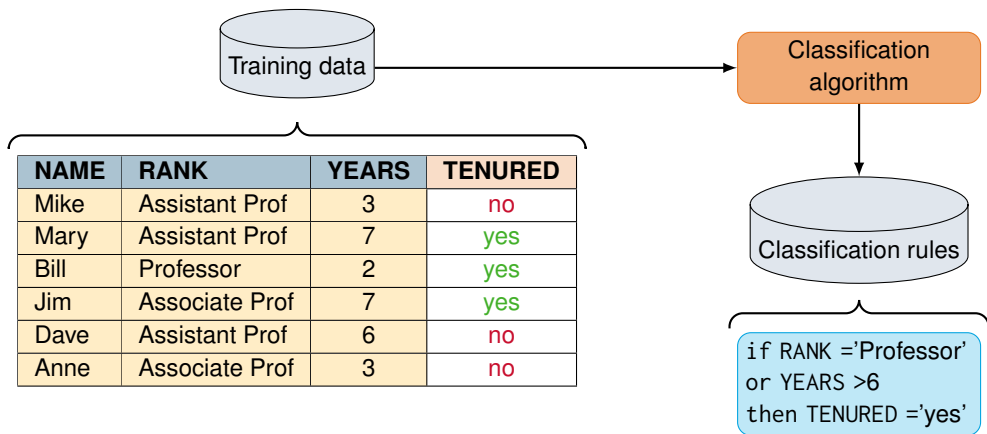
## 1. Model construction: describing a set of predetermined classes:

- Each tuple/sample is assumed to belong to a predefined class, as determined by the **class-label attribute**.
- The set of tuples used for model construction is the **training set**.
- The **model** is represented as classification rules, decision trees, or mathematical formulae.

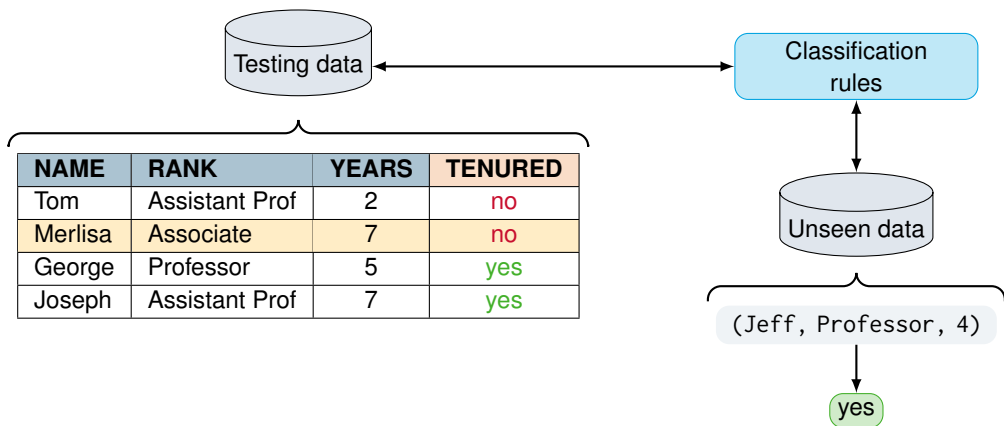
## 2. Model usage, for classifying future or unknown objects:

- Estimate **accuracy** of the model:
  - The known label of **test samples** is compared with the result from the model.
  - **Accuracy rate** is the percentage of test-set samples that are correctly classified by the model.
  - Test set is independent of training set (otherwise overfitting).
  - More on evaluation metrics later in [▶ section "Model Evaluation and Selection"](#).
- If the accuracy is acceptable, **use the model** to classify data tuples whose class labels are not known.

## Classification – A Two-step Process: 1. Model Construction



## Classification – A Two-step Process: 2. Model Usage





---

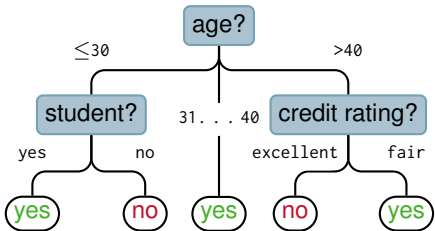
# Decision Tree Induction

## Decision Tree: An Example

- **Training dataset:**  
**buys\_computer.**

The dataset follows an example of Quinlan's ID3.

- **Resulting tree:**



age	income	student	credit_rating	buys_computer
≤ 30	high	no	fair	no
≤ 30	high	no	excellent	no
31 ... 40	high	no	fair	yes
> 40	medium	no	fair	yes
> 40	low	yes	fair	yes
> 40	low	yes	excellent	no
31 ... 40	low	yes	excellent	yes
≤ 30	medium	no	fair	no
≤ 30	low	yes	fair	yes
> 40	medium	yes	fair	yes
≤ 30	medium	yes	excellent	yes
31 ... 40	medium	no	excellent	yes
31 ... 40	high	yes	fair	yes
> 40	medium	no	excellent	no

# Decision Tree

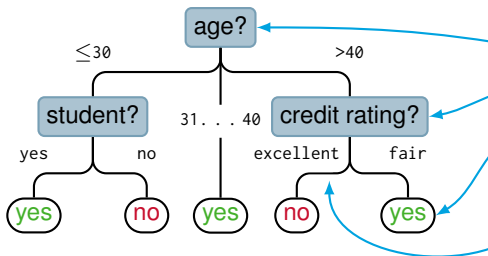
## Decision Tree Induction

*Decision tree induction* refers to the learning of a decision-tree based on labeled training data.

## Decision Tree

A *decision tree* is a flowchart-like structure consisting of interconnected internal and leaf nodes.

### Components of a Decision Tree



- **Root:** topmost node.
- **Internal node:** test on an attribute.
- **Leaf node:** holds a class label, also called *terminal node*.
- **Branch:** outcome of a leaf node's test coupled with a text. In this example: excellent.

## Algorithm for Decision Tree Induction (I)

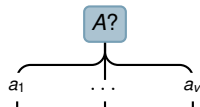
**Construction in general** follows a greedy algorithm, i. e. non-backtracking. Thus, it is done in a *top-down recursive in a divide-and-conquer manner*.

**Input:** data partition  $D$ , `attribute_list`, `attribute_selection_method`.

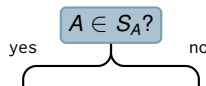
**Algorithm Sketch `build_decision_tree`:**

1. Create node  $N$ .
2. Determine splitting attribute  $A$  according to the splitting criterion obtained by applying `attribute_selection_method`. May also return a *split point* or *splitting subset*.
3. Label  $N$  with splitting criterion.
4. If splitting attribute is discrete-valued and multiway split is allowed, or attribute has only one unique value: Remove attribute from `attribute_list`.
5. For each outcome of splitting criterion:
  - Partition  $D$  according to outcome of splitting criterion.
  - Grow branches (subtrees, call `build_decision_tree`) on  $N$  for each partition.
6. Return node  $N$

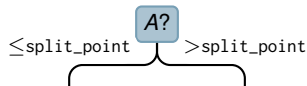
Attribute Types:  
Discrete:



Discrete & Binary Tree:



Continuous:



## Algorithm for Decision Tree Induction (II)

### Stopping criteria:

- All samples in  $D$  belong to the same class.  $N$  becomes a leaf.
- `attribute_list` is empty. If multiple classes: use majority class.
- Partition  $D$  is empty, thus create leaf with majority class.

### Decision Tree Algorithm

We merely discussed the gist to build a decision tree. A detailed algorithm to construct a decision tree is covered in the appendix under [▶ section "Basic Decision Tree Algorithm"](#), as well as in our reference book<sup>1</sup>.

<sup>1</sup>J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011, ISBN: 0123814790, pp. 332 – 335.

# Attribute Selection Methods

## Attribute Selection Methods

An *attribute selection method* is a heuristic to determine the “best” splitting criterion to partition data.

- Also known as *splitting rules*.
- Provides ranking for each attribute.
- Partition data based on attribute with best score. *Best score* depends on method used (some seek to minimize whereas others maximize).
- Tree node is labeled with splitting criterion (attribute). Sub tree results from partitions of splitting criterion.

Popular methods include:

1. Information Gain
2. Gain Ratio
3. Gini Index

## Attribute Selection Methods: Information Gain (ID3) (I)

- **Select the attribute with the highest information gain.**
- Partitions reflect least randomness, i. e. impurity.
- **Expected information** (entropy) needed to classify a tuple in  $D$ :

$$\text{Info}(D) = - \sum_{i=1}^m p_i \log_2(p_i)$$

- $p_i$  is the probability that tuple in  $D$  belongs to class  $C_i$ ,  
estimated by  $\frac{|C_i|}{|D|}$ , such that  $1 \leq i \leq m$ .
- $\log_2$  because information is encoded in bits.

## Attribute Selection Methods: Information Gain (ID3) (II)

Calculate information for every attribute in `attribute_list` and data partition  $D$ :

### Discrete-valued Attribute

- Attribute  $A$  with  $v$  distinct values.
- Expected information required to classify tuple in  $D$  based on partitioning by  $A$ .
- $D_A$ : dataset  $D$  partitioned by  $A$ ,  $v$ : number of distinct values of  $A$ .

$$\text{Info}_A(D) = \sum_{j=1}^v \frac{|D_{A,j}|}{|D_A|} \text{Info}(D_{A,j})$$

### Continuous-valued Attribute

- Attribute  $A$  with  $v$  distinct values.
- Order values in increasing order.
- Calculate midpoint of every neighbouring value  $\frac{a_i + a_{i+1}}{2}$ . Results in  $v - 1$  possible split points.
- Evaluate  $\text{Info}_A(D)$  for every possible splitting.
- Binary split:  $A \leq \text{split\_point}$  and  $A > \text{split\_point}$

Given  $\text{Info}(D)$  and  $\text{Info}_A(D)$ , Information Gain is defined as:

$$\text{Gain}(A) = \text{Info}(D) - \text{Info}_A(D)$$



## Example: Information Gain

- **Class P:** buys\_computer = "yes"
- **Class N:** buys\_computer = "no"

$$\text{Info}(D) = I(9, 5) = -\frac{9}{14} \log_2\left(\frac{9}{14}\right) - \frac{5}{14} \log_2\left(\frac{5}{14}\right) = 0.94$$

age	p	n	$I(p, n)$
$\leq 30$	2	3	0.971
31 ... 40	4	0	0
$> 40$	3	2	0.971

- **Similarly,**
  - $\text{Gain}(\text{income}) = 0.029$ ,
  - $\text{Gain}(\text{student}) = 0.151$ ,
  - $\text{Gain}(\text{credit\_rating}) = 0.048$ .

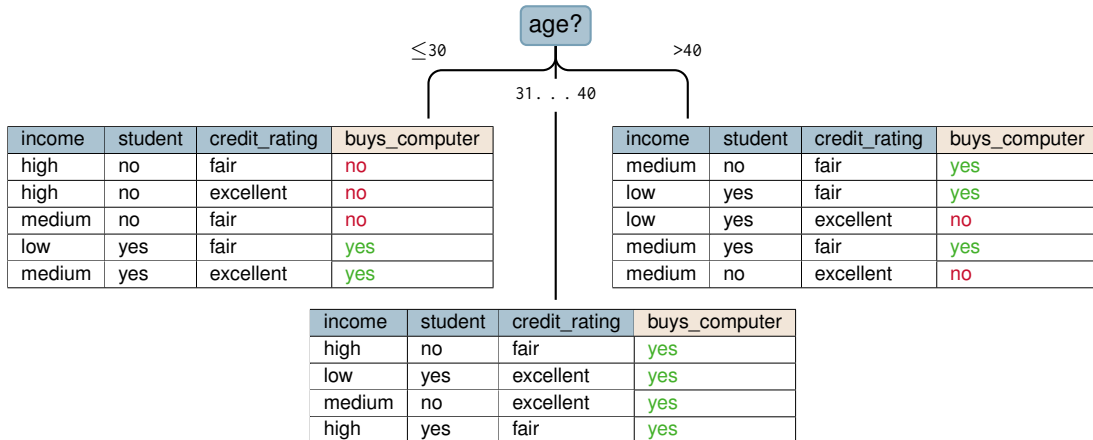
$$\text{Info}_{\text{age}}(D) = \frac{5}{14} I(2, 3) + \frac{4}{14} I(4, 0) + \frac{5}{14} I(3, 2) = 0.694.$$

$\frac{5}{14} I(2, 3)$  means "age  $\leq 30$ " has 5 out of 14 samples, with 2 yes'es and 3 no's. Hence,

$$\text{Gain}(\text{age}) = \text{Info}(D) - \text{Info}_{\text{age}}(D) = 0.246.$$

age	income	student	credit_rating	buys_computer
$\leq 30$	high	no	fair	no
$\leq 30$	high	no	excellent	no
31 ... 40	high	no	fair	yes
$> 40$	medium	no	fair	yes
$> 40$	low	yes	fair	yes
$> 40$	low	yes	excellent	no
31 ... 40	low	yes	excellent	yes
$\leq 30$	medium	no	fair	no
$\leq 30$	low	yes	fair	yes
$> 40$	medium	yes	fair	yes
$\leq 30$	medium	yes	excellent	yes
31 ... 40	medium	no	excellent	yes
31 ... 40	high	yes	fair	yes
$> 40$	medium	no	excellent	no

## Partitioning in the Example



## Attribute Selection Methods: Gain Ratio (C4.5)

- Extension to Information Gain as this method is biased towards attributes with large amount of values.
- **C4.5 (a successor of ID3) uses gain ratio to overcome the problem (normalization to information gain):**

$$\text{SplitInfo}_A(D) = - \sum_{j=1}^v \frac{|D_j|}{|D|} \log_2 \left( \frac{|D_j|}{|D|} \right),$$

$$\text{GainRatio}(A) = \frac{\text{Gain}(A)}{\text{SplitInfo}_A(D)}.$$

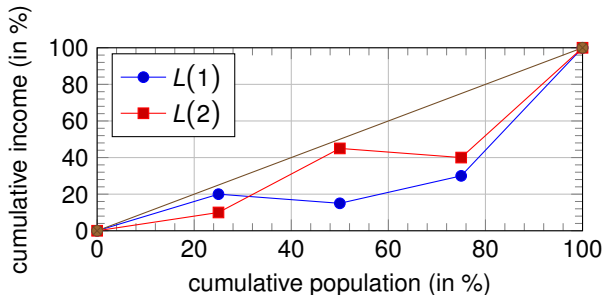
- The attribute with the maximum gain ratio is selected as the splitting attribute.
- **Disadvantage:** Becomes unstable as  $\text{SplitInfo}_A(D)$  approaches zero. Overcome by using Information Gain then.

## Gini Index

- **Corrado Gini (1884 – 1965).**
  - Italian statistician and sociologist.
- **Also called Gini coefficient.**
- **Measures statistical dispersion.**
  - Zero expresses perfect equality where all values belong to the same class.
  - One expresses maximal inequality among values.
- **Based on the Lorenz curve.**
  - Plots the proportion of the total sum of values ( $y$ -axis) that is cumulatively assigned to the bottom  $x\%$  of the population.
  - Line at 45 degrees thus represents perfect equality of value distribution.
- **Gini coefficient then is . . .**
  - . . . the ratio of the area that lies between the line of equality and the Lorenz curve over the total area under the line of equality.

## Gini Index (II)

Example: Distribution of incomes.



## Gini Index (CART, IBM IntelligentMiner) (I)

- Measured impurity of partition  $D$  is defined as the sum over  $n$  classes:

$$\text{Gini}(D) = 1 - \sum_{j=1}^n p_j^2,$$

where  $p_j$  is the non-zero probability that sample in  $D$  belongs to class  $C_j$  as estimated by  $\frac{|C_j, D|}{|D|}$

- If **attribute  $A$  is discrete-valued** with  $v$  distinct values compute all possible subsets of values  $2^v - 2$ . Compute weighted sum of each partition tuple ( $D_1$  and  $D_2$ ) as follows:

$$\text{Gini}_A(D) = \frac{|D_1|}{|D|} \text{Gini}(D_1) + \frac{|D_2|}{|D|} \text{Gini}(D_2).$$

- If **attribute  $A$  is continuous-valued** proceed similarly as in calculating Information Gain for continuous-valued attributes (order values, calculate midpoint of value pairs) and then calculate  $\text{Gini}_A(D)$  for every split point.

## Gini Index (CART, IBM IntelligentMiner) (II)

- Gini Index as the **reduction in impurity** is then given as follows:

$$\Delta \text{Gini}_A(D) = \text{Gini}(D) - \text{Gini}_A(D).$$

- Attribute with minimum Gini Index is used as the splitting attribute.

## Example: Gini Index (I)

- $D$  has 9 tuples in `buys_computer = "yes"` and 5 in "no", thus

$$\text{Gini}(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459.$$

- Suppose the attribute `income` partitions  $D$  into 10 in  $D_1 : \{\text{low}, \text{medium}\}$  and 4 in  $D_2 : \{\text{high}\}$ :

$$\begin{aligned} & \text{Gini}(D|_{D[\text{income}] = \text{"medium"}, \text{"low"}}) \\ &= \frac{10}{14} \text{Gini}(D_1) + \frac{4}{14} \text{Gini}(D_2) \\ &= \frac{10}{14} \left( 1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2 \right) + \frac{4}{14} \left( 1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2 \right) = \\ &= 0.443 = \text{gini}(D|_{D[\text{income}] = \text{"high"}}). \end{aligned}$$



## Computation of Gini Index (II)

- $\text{Gini}(D|_{D[\text{income}] = \text{"low"}, \text{"high"}}) = 0.458,$   
 $\text{Gini}(D|_{D[\text{income}] = \text{"medium"}, \text{"high"}}) = 0.450.$
- Thus, split on the {"low","medium"} and {"high"}, since it has the lowest gini index.

# Attribute Selection Methods Overview

The three methods, in general, return good results, but

- **Information gain:**
  - Biased towards multi-valued attributes.
- **Gain ratio:**
  - Tends to prefer unbalanced splits in which one partition is much smaller than the others.
- **Gini index:**
  - Biased to multi-valued attributes.
  - Has difficulty when number of classes is large.
  - Tends to favor tests that result in equal-sized partitions and purity in both partitions.

## Other Attribute Selection Methods

- **CHAID:**
  - A popular decision tree algorithm, measure based on  $\chi^2$  test for independence.
- **C-SEP:**
  - Performs better than Information Gain and Gini Index in certain cases.
- **G-statistic:**
  - Has a close approximation to  $\chi^2$  distribution.
- **MDL (Minimal Description Length) principle:**
  - I.e. the simplest solution is preferred.
  - The best tree is the one that requires the fewest number of bits to both (1) encode the tree and (2) encode the exceptions to the tree.
- **Multivariate splits:**
  - Partitioning based on multiple variable combinations.
  - CART: finds multivariate splits based on a linear combination of attributes.
- **Which Attribute Selection Method is the best?**
  - Most give good results, none is significantly superior to others.

# Overfitting and Tree Pruning

- **Overfitting: An induced tree may overfit the training data.**
  - Too many branches, some may reflect anomalies due to noise or outliers.
  - Poor accuracy for unseen samples.
- Pruned trees are typically smaller, less complex, easier to comprehend, faster and better at classifying unseen data.
- **Two approaches to avoid overfitting:**
  1. **Prepruning:**
    - Halt tree construction early.  
Do not split a node, if this would result in the goodness measure falling below a threshold.
    - Difficult to choose an appropriate threshold.
  2. **Postpruning:**
    - 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."

## Enhancements to Basic Decision Tree Induction

- **Allow for continuous-valued attributes.**
  - Dynamically define new discrete-valued attributes that partition the values of continuous-valued attributes into a discrete set of intervals.
- **Handle missing attribute values.**
  - Assign the most common value of the attribute.
  - Assign probability to each of the possible values.
- **Attribute construction.**
  - Create new attributes based on existing ones that are sparsely represented.
  - This reduces fragmentation, repetition, and replication.

# Classification in Large Databases

- ID3, C4.5, and CART have been developed with the assumption that data fits into memory. With Big Data that's not possible anymore.
- **Scalability:**
  - Classifying datasets with millions of examples and hundreds of attributes with reasonable speed.
- **Why is decision tree induction popular?**
  - Relatively fast learning speed (compared to other classification methods).
  - Convertible to simple and easy-to-understand classification rules.
  - Can use SQL queries for accessing databases.
  - Classification accuracy comparable with other methods.
- Two scalable methods, among others:
  1. RainForest
  2. BOAT

## Scalable Decision Tree: RainForest

- Applicable to any decision tree algorithm.
- **Separates the scalability aspects from the criteria that determine the quality of the tree.**
- **Builds an AVC-list:** (Attribute, Value, Class\_label).
- **AVC-set (of an attribute X):**
  - Projection of training dataset onto the attribute  $X$  and class label where counts of individual class label are aggregated.
- **AVC-group (of a node  $n$ ):**
  - Set of AVC-sets of all predictor attributes at the node  $n$ .

## RainForest: Training Set and its AVC-sets

age	income	student	credit_rating	buys_computer
$\leq 30$	high	no	fair	no
$\leq 30$	high	no	excellent	no
31 ... 40	high	no	fair	yes
$> 40$	medium	no	fair	yes
$> 40$	low	yes	fair	yes
$> 40$	low	yes	excellent	no
31 ... 40	low	yes	excellent	yes
$\leq 30$	medium	no	fair	no
$\leq 30$	low	yes	fair	yes
$> 40$	medium	yes	fair	yes
$\leq 30$	medium	yes	excellent	yes
31 ... 40	medium	no	excellent	yes
31 ... 40	high	yes	fair	yes
$> 40$	medium	no	excellent	no

AVC-set on age:

age	yes	no
$\leq 30$	2	3
31 ... 40	4	0
$> 40$	3	2

AVC-set on income:

income	yes	no
high	2	2
medium	4	2
low	3	1



## RainForest: Training Set and its AVC-sets (II)

age	income	student	credit_rating	buys_computer
$\leq 30$	high	no	fair	no
$\leq 30$	high	no	excellent	no
31 ... 40	high	no	fair	yes
$> 40$	medium	no	fair	yes
$> 40$	low	yes	fair	yes
$> 40$	low	yes	excellent	no
31 ... 40	low	yes	excellent	yes
$\leq 30$	medium	no	fair	no
$\leq 30$	low	yes	fair	yes
$> 40$	medium	yes	fair	yes
$\leq 30$	medium	yes	excellent	yes
31 ... 40	medium	no	excellent	yes
31 ... 40	high	yes	fair	yes
$> 40$	medium	no	excellent	no

AVC-set on student:

student	yes	no
yes	6	1
no	3	4

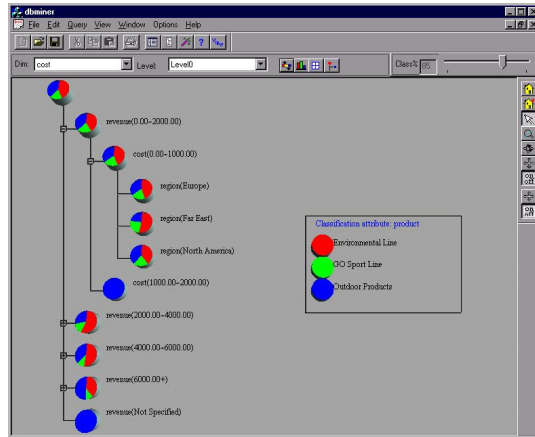
AVC-set on credit\_rating:

credit_rating	yes	no
fair	6	2
excellent	3	3

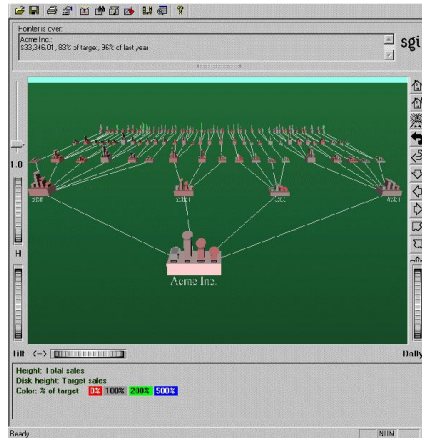
## Scalable Decision Tree: BOAT

- BOAT = Bootstrapped Optimistic Algorithm for Tree Construction
- **Use a statistical technique called bootstrapping to create several smaller samples (subsets), each fitting in memory.**
  - See on the subsequent slides.
- **Each subset is used to create a tree, resulting in several trees.**
- **These trees are examined and used to construct a new tree  $T'$ .**
  - It turns out that  $T'$  is very close to the tree that would be generated using the whole data set together.
- **Advantages:**
  - Requires only two scans of DB.
  - An incremental algorithm:
    - Take insertions and deletions of training data and update the decision tree.

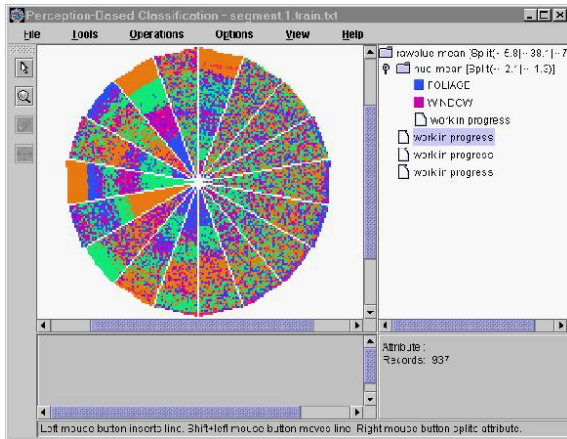
# Presentation of Classification Results



# Visualization of a Decision Tree in SGI/MineSet 3.0



# Interactive Visual Mining by Perception-based Classification (PBC)



---

# Rule-Based Classification

## Using IF-THEN Rules for Classification

- **Represent the knowledge in the form of IF-THEN rules.**
  - E.g., if age  $\leq 30$  AND student = "yes" THEN buys\_computer = "yes".
  - Readable.
- **Rule antecedent/precondition vs. rule consequent.**
- **Assessment of a rule  $R$ : coverage and accuracy.**
  - $n_{\text{covers}} = \#$  of tuples covered by  $R$  (antecedent if true).
  - $n_{\text{correct}} = \#$  of tuples correctly classified by  $R$ .
  - $\text{coverage}(R) = \frac{n_{\text{covers}}}{|D|}$  with  $D$  training data set.
  - $\text{accuracy}(R) = \frac{n_{\text{correct}}}{n_{\text{covers}}}$ .

## Potential Problems of Rule-Based Classification

1. More than one rule is triggered.
2. No rule is triggered.



# Potential Problems of Rule-based Classification: Solutions

## 1. More than one rule is triggered: **conflict resolution**.

- **Size ordering:**
  - Assign the highest priority to the triggered rule that has the "toughest" requirement (i.e., rule with most used attribute in condition).
- **Class-based ordering:**
  - Decreasing order of prevalence or misclassification cost per class.
  - No order of rules within class → disjunction (logical OR) between rules.
- **Rule-based ordering** (decision list):
  - Rules are organized into one long priority list, according to some measure of rule quality, or by experts.
  - Rules must be applied in this particular order to avoid conflict.

## 2. No rule is triggered.

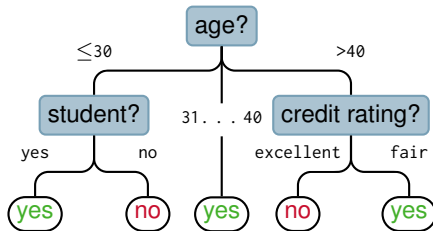
- Use a fallback/default rule.
- Always evaluated as the last rule, if and only if other rules are not covered by some tuple, i. e. no rules have been triggered.

## Rule Extraction from a Decision Tree

- Rules are **easier to understand** than large trees.
- Rule can be created for **each path from the root to a leaf**.
  - The leaf holds the class prediction.
- Each attribute-value pair along the path forms a conjunction:

### Example:

1. IF  $\text{age} \leq 30$  AND  $\text{student} = \text{"no"}$   
THEN  $\text{buys\_computer} = \text{"no"}$ .
2. IF  $\text{age} \leq 30$  AND  $\text{student} = \text{"yes"}$   
THEN  $\text{buys\_computer} = \text{"yes"}$ .
3. IF  $\text{age} == 31 \dots 40$  THEN  $\text{buys\_computer} = \text{"yes"}$ .
4. ...



## Rule Induction: Sequential Covering Method

- **Sequential covering algorithm:**
  - Extracts rules directly from training data.
- **Typical sequential covering algorithms:**
  - FOIL, AQ, CN2, RIPPER.
- **Rules are learned sequentially.**
  - Each rule for a given class  $C_i$  will cover many tuples of  $C_i$ , but none (or few) of the tuples of other classes.
- **Algorithm sketch:**
  - Rules are learned one at a time.
  - Each time a rule is learned, the tuples covered by the rule are removed.
  - The process repeats on the remaining tuples unless termination condition, e.g., when no more training examples left or when the quality of a rule returned is below a user-specified threshold.
- **Compare with decision-tree induction:**
  - That was learning a set of rules simultaneously.

## Sequential Covering Algorithm (I): Basic Algorithm

### Data:

- Training dataset  $D$  containing tuples with their associated class labels;
- `attribute_values`, the set of all attributes and their possible values;

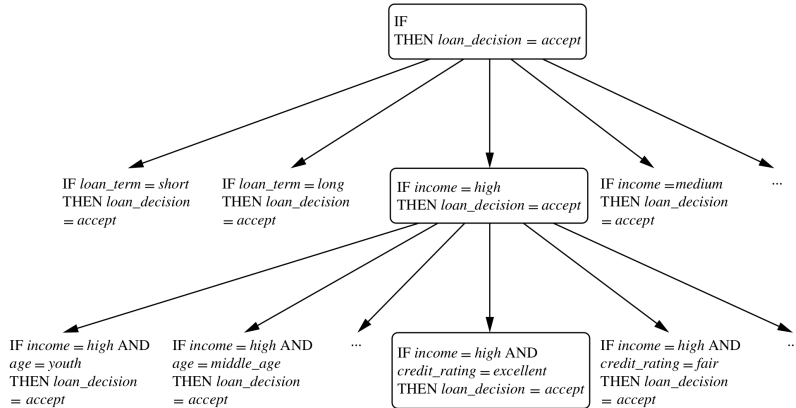
### Result: A rule set.

```
1 rule_set  $\leftarrow$  {} ; // Initial set of rules learned is empty
2 foreach class  $c$  of  $D$  do
3     repeat
4         rule  $\leftarrow$  learn_one_rule( $D$ , attribute_values,  $c$ );
5         remove tuples covered by rule from  $D$ ;
6         rule_set  $\leftarrow$  rule_set + rule ; // add new rule to rule set
7     until terminating condition;
8 return rule_set;
```

## Sequential Covering Algorithm (II): How Rules are Learned

- Rules are learned in a *general-to-specific* manner
- Start with the most general rule possible: a rule with an empty condition  
IF THEN buys\_computer = "yes"
- Then: Consider each possible attribute (attribute\_values)  
For instance: attribute-value pair (att, val); consider the following attribute tests:  
att = val, att < val, att ≤ val, att > val, att ≥ val
- **Curse of dimensionality:** testing each attribute-value pair is computationally explosive
- Solution: **greedy depth-first strategy of learn\_one\_rule**
  - Add new attribute test that improves the rule quality the most.
  - Each time a new attribute test is added, the rule should cover more "accept" tuples (buys\_computer = "yes").
  - Repeat until a certain acceptable quality level is reached (*terminating condition*).
- **What if we added a poor choice?** Greedy search does not allow for backtracking.
  - Retain the best  $k$  attribute candidates at each step, rather than a single best candidate.

## Sequential Covering Algorithm (III): Rule Space



## Sequential Covering Algorithm (III): Rule Quality Measure

- `learn_one_rule` requires a measure of rule quality.
- Accuracy and coverage seems obvious choices on their own, but individually not enough.
- **FOIL** (First-Order Inductive Learner): based on information gain
  - Suppose we have two rules:

$R$  : IF condition THEN class =  $c$

$R'$  : IF condition' THEN class =  $c$

- $pos/neg$  are # of positive/negative tuples covered by  $R$ ,  $pos'/neg'$  respectively for  $R'$ .
- FOIL assesses the information gained by extending  $condition'$  as

$$\text{FOIL\_Gain} = pos' \left( \log_2 \frac{pos'}{pos' + neg'} - \log_2 \frac{pos}{pos + neg} \right).$$

- FOIL favors rules that have high accuracy and cover many positive tuples.

## Rule Pruning

- **Danger of overfitting.**
- **Removing a conjunct (attribute test),**
  - if pruned version of rule has greater quality, assessed on an independent set of test tuples (called "pruning set").
- **FOIL uses:**

$$\text{FOIL\_Prune}(R) = \frac{\text{pos} - \text{neg}}{\text{pos} + \text{neg}}.$$

- If FOIL\_Prune is higher for the pruned version of  $R$ , prune  $R$ .



---

# Bayes Classification Methods

# Bayesian Classification: Why?

- **A statistical classifier:**
  - Performs probabilistic prediction, i.e. predicts class-membership probabilities.
- **Foundation:** Bayes' Theorem.
- **Performance:**
  - A simple Bayesian classifier (naïve Bayesian classifier) has performance comparable with decision tree and selected neural-network classifiers.
- **Incremental:**
  - Each training example can incrementally increase/decrease the probability that a hypothesis is correct – prior knowledge can be combined with observed data.
- **Standard:**
  - Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured.

## Bayes' Theorem: Basics

- **Let  $X$  be a data sample ("evidence").**
  - The class label shall be unknown.
- **Let  $C_i$  be the hypothesis that  $X$  belongs to class  $i$ .**
- **Classification is to determine  $P(C_i|X)$ :**
  - **Posteriori probability:** the probability that the hypothesis holds given the observed data sample  $X$ .
- $P(C_i)$ :
  - **Prior probability:** the initial probability.
  - E.g.,  $X$  will buy computer, regardless of age, income, . . .
- $P(X)$ :
  - Probability that sample data is observed.
- $P(X|C_j)$ :
  - **Likelihood:** the probability of observing the sample  $X$  given that the hypothesis holds.
  - E.g., given that  $X$  buys computer, the probability that  $X$  is 31 . . . 40, medium income.

## Bayes' Theorem: Basics (II)

- **Given training data  $X$ , the posteriori probability  $P(C_i|X)$  of a hypothesis  $C_i$  follows from the Bayes' Theorem:**

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}.$$

- **Predicts that  $X$  belongs to  $C_i$  if the probability  $P(C_i|X)$  is **the highest** among all the  $P(C_k|X)$  for all  $k$  classes.**
- **Practical difficulty:**
  - Requires initial knowledge of many probabilities.
  - Significant computational cost.

## Towards Naïve Bayesian Classifier

- Let  $D$  be a training set of tuples and their associated class labels.
  - Each tuple is represented by an  $n$ -dimensional attribute  $X = (x_1, x_2, \dots, x_n)$ .
- Suppose there are  $m$  classes  $C_1, C_2, \dots, C_m$ .
- Classification is to derive the **maximum posteriori probability**.
  - i.e. the maximal  $P(C_i|X)$ .
- This can be derived from Bayes' Theorem:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}.$$

- Since  $P(X)$  is constant for all classes, we must maximize only:

$$P(X|C_i)P(C_i).$$

## Derivation of Naïve Bayes Classifier

- **A simplifying assumption: attributes are conditionally independent.**
  - I.e. no dependence relation between attributes (which is "naïve").

$$P(X|C_i) = \prod_{k=1}^n P(x_k|C_i) = P(x_1|C_i)P(x_2|C_i) \cdots P(x_n|C_i).$$

- This greatly reduces the computation cost:  
Only count the class distribution.
- If  $A_k$  is categorical,
  - $P(x_k|C_i)$  is the number of tuples in  $C_i$  having value  $x_k$  for  $A_k$  divided by  $|C_{i,D}|$  (the number of tuples of  $C_i$  in  $D$ ).
- If  $A_k$  is continuous-valued,
  - $P(x_k|C_i)$  is usually computed based on Gaussian distribution with a mean  $\mu$  and standard deviation  $\sigma$ :

$$G(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

- and  $P(x_k|C_i) = G(x_k, \mu_{C_i}, \sigma_{C_i})$ .

# Naïve Bayesian Dataset

- Classes:**

- $C_1$ : buys\_computer = "yes".
- $C_2$ : buys\_computer = "no".

- Data sample:**

- $X = (\text{age} \leq 30, \text{income} = \text{"medium"}, \text{student} = \text{"yes"}, \text{credit\_rating} = \text{"fair"})$ .

age	income	student	credit_rating	buys_computer
$\leq 30$	high	no	fair	no
$\leq 30$	high	no	excellent	no
31 ... 40	high	no	fair	yes
$> 40$	medium	no	fair	yes
$> 40$	low	yes	fair	yes
$> 40$	low	yes	excellent	no
31 ... 40	low	yes	excellent	yes
$\leq 30$	medium	no	fair	no
$\leq 30$	low	yes	fair	yes
$> 40$	medium	yes	fair	yes
$\leq 30$	medium	yes	excellent	yes
31 ... 40	medium	no	excellent	yes
31 ... 40	high	yes	fair	yes
$> 40$	medium	no	excellent	no

## Naïve Bayesian Classifier: An Example

- Prior probability  $P(C_i)$ :
  - $P(\text{buys\_computer} = \text{"yes"}) = \frac{9}{14} = 0.643$ .
  - $P(\text{buys\_computer} = \text{"no"}) = \frac{5}{14} = 0.357$ .
- New tuple  $X = (\text{age} \leq 30, \text{income} = \text{"medium"}, \text{student} = \text{"yes"}, \text{credit\_rating} = \text{"fair"})$ .
- **Compute likelihood  $P(X|C_i)$  for each class:**
  - $P(\text{age} \leq 30 | \text{buys\_computer} = \text{"yes"}) = \frac{2}{9} = 0.222$ .
  - $P(\text{age} \leq 30 | \text{buys\_computer} = \text{"no"}) = \frac{3}{5} = 0.6$ .
  - $P(\text{income} = \text{"medium"} | \text{buys\_computer} = \text{"yes"}) = \frac{4}{9} = 0.444$ .
  - $P(\text{income} = \text{"medium"} | \text{buys\_computer} = \text{"no"}) = \frac{2}{5} = 0.4$ .
  - $P(\text{student} = \text{"yes"} | \text{buys\_computer} = \text{"yes"}) = \frac{5}{9} = 0.556$ .
  - $P(\text{student} = \text{"yes"} | \text{buys\_computer} = \text{"no"}) = \frac{1}{5} = 0.2$ .
  - $P(\text{credit\_rating} = \text{"fair"} | \text{buys\_computer} = \text{"yes"}) = \frac{6}{9} = 0.667$ .
  - $P(\text{credit\_rating} = \text{"fair"} | \text{buys\_computer} = \text{"no"}) = \frac{2}{5} = 0.4$ .



## Naïve Bayesian Classifier: An Example (II)

- Compute likelihood  $P(X|C_i)$  for this new tuple  $X$ :
  - Recall:  $P(X|C_i) = \prod_{k=1}^n P(x_k|C_i) = P(x_1|C_i)P(x_2|C_i) \cdots P(x_n|C_i)$ .
  - $P(X|\text{buys\_computer} = \text{"yes"}) = 0.222 \cdot 0.444 \cdot 0.556 \cdot 0.667 = 0.037$ .
  - $P(X|\text{buys\_computer} = \text{"no"}) = 0.6 \cdot 0.4 \cdot 0.2 \cdot 0.4 = 0.019$ .
- Compute  $P(X|C_i) \cdot P(C_i)$  for each class outcome:
  - $P(X|\text{buys\_computer} = \text{"yes"}) \cdot P(\text{buys\_computer} = \text{"yes"}) = 0.024$ .
  - $P(X|\text{buys\_computer} = \text{"no"}) \cdot P(\text{buys\_computer} = \text{"no"}) = 0.007$ .
- **Therefore,  $X$  belongs to class  $C_1$  ( $\text{buys\_computer} = \text{"yes"}$ ) with probability of 2.4%.**

## Avoiding the Zero-Probability Problem

- **Naïve Bayesian prediction requires each conditional probability to be non-zero.**
  - Otherwise, the predicted probability will be zero.

$$P(X|C_i) = \prod_{k=1}^n P(x_k|C_i).$$

- **Example:**

- Suppose a dataset with 1000 tuples, income = "low" (0), income = "medium" (990), and income = "high" (10).

- **Use Laplacian correction (or Laplacian estimator):**

- Add 1 to each case:
    - $P(\text{income} = \text{"low"}) = \frac{1}{1003}.$
    - $P(\text{income} = \text{"medium"}) = \frac{991}{1003}.$
    - $P(\text{income} = \text{"high"}) = \frac{11}{1003}.$
  - Additionally, add 1 to *each* other attribute and value combination as well!
  - The "corrected" probability estimates are close to their "uncorrected" counterparts.

# Naïve Bayesian Classifier: Comments

## Advantages

- Easy to implement.
- Good results obtained in most of the cases.

## Disadvantages

- Assumption: class conditional independence, therefore loss of accuracy.
- Practically, **dependencies** exist among variables.
  - E.g., hospital patients:
    - Profile: age, family history, etc.
    - Symptoms: fever, cough, etc.
    - Disease: lung cancer, diabetes, etc.
  - Cannot be modeled by naïve Bayesian classifier.

**How to deal with these dependencies?** → Bayesian belief networks (see textbook).

---

# Model Evaluation and Selection

# Model Evaluation and Selection

- **Evaluation metrics:**
  - How can we measure accuracy?
  - Other metrics to consider?
- Use **test** set of class-labeled tuples instead of training set when assessing accuracy.
- **Methods for estimating a classifier's accuracy:**
  - Holdout method, random subsampling.
  - Cross-validation.
  - Bootstrap.
- **Comparing classifiers:**
  - Confidence intervals.
  - Cost-benefit analysis and ROC curves.

## Confusion Matrix and Evaluation Metrics

Given  $M$  classes, an entry  $C_{ij}^{(m)}$  in an  $M \times M$  confusion matrix indicates the number of tuples in class  $i$  that were labeled by the classifier as class  $j$ .

		Predicted Class		Total
		$C_1$	$\neg C_1$	
True Class	$C_1$	<b>TP</b>	<b>FN</b>	<b>P</b>
	$\neg C_1$	<b>FP</b>	<b>TN</b>	<b>N</b>
Total		<b>P'</b>	<b>N'</b>	<b>P + N</b>

- **True Positives (TP)** = correctly classified tuples.
- **True Negatives (TN)** = correctly classified tuples.
- **False Positives (FP)** = negative tuples incorrectly classified as positive.
- **False Negatives (FN)** = positive tuples incorrectly classified as negative.

### Accuracy:

- Percentage of correctly classified tuples.
- Also known as the (overall) recognition rate.
- Most effective with a *balanced dataset*.
- Inverse: **Error rate** as the misclassification rate.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}}$$

$$\begin{aligned} \text{Error Rate} &= 1 - \text{Accuracy} \\ &= \frac{\text{FP} + \text{FN}}{\text{P} + \text{N}} \end{aligned}$$

## Confusion Matrix and Evaluation Metrics (II)

		Predicted Class		Total
		$C_1$	$\neg C_1$	
True Class	$C_1$	<b>TP</b>	<b>FN</b>	<b>P</b>
	$\neg C_1$	<b>FP</b>	<b>TN</b>	<b>N</b>
Total		<b>P'</b>	<b>N'</b>	<b>P + N</b>

- **True Positives (TP)** = correctly classified tuples.
- **True Negatives (TN)** = correctly classified tuples.
- **False Positives (FP)** = negative tuples incorrectly classified as positive.
- **False Negatives (FN)** = positive tuples incorrectly classified as negative.

- **Sensitivity** = True positive rate.
- **Specificity** = True negative rate.
- **Precision** = Measure of exactness.
- **Recall** = Measure of completeness.  
Perfect score is 1.0.  
Inverse relationship with precision.

$$\text{Sensitivity} = \frac{TP}{P} = \frac{TP}{TP + FN} = \text{Recall}$$

$$\text{Specificity} = \frac{TN}{N}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

## Confusion Matrix and Evaluation Metrics (III)

**F-Measure:** Combines precision and recall in one single measure.

### $F_1$ Measure

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Harmonic mean between precision and recall.
- Equal weight to both measures.

### $F_\beta$ Measure

$$F_\beta = \frac{(1 + \beta^2) \times \text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}}$$

- Weighted measure.
- Gives  $\beta$ -times more weight to precision.
- $\beta > 1$ : More weight on precision. E.g. important to minimize false positives.
- $\beta < 1$ : More weight on recall. E.g. important to minimize false negatives.



## Example: Confusion Matrix and Evaluation Metrics

Actual class/predicted class	cancer = yes	cancer = no	Total	Recognition (%)
cancer = yes	<b>90</b>	<b>210</b>	300	30.00 (sensitivity)
cancer = no	<b>140</b>	<b>9560</b>	9700	98.56 (specificity)
Total	230	9770	10000	96.40 (accuracy)

- Precision =  $\frac{90}{230} = 39.13\%$ .
- Recall =  $\frac{90}{300} = 30.00\%$ .
- $F_1$ -measure =  $\frac{2 \cdot 0.3913 \cdot 0.3}{0.3913 + 0.3} = 33.96\%$ .

# Evaluation Strategies: Holdout Method

## Holdout method.

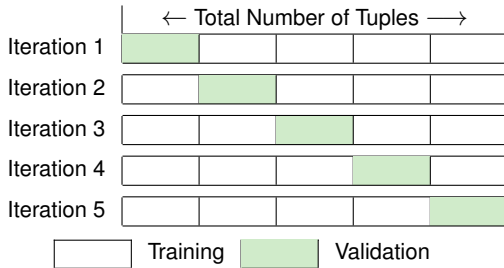
- Randomly assign tuples into two independent sets:
  - **Training set** (E.g.,  $2/3$ ) for model construction.
  - **Test set** (E.g.,  $1/3$ ) for accuracy estimation.
- Random sampling: a variation of holdout that repeats holdout  $k$  times.
  - Create an average accuracy over all experiments.

## Evaluation Strategies: Cross Validation

Most common:  $k$ -fold cross validation ( $k = 10$  is popular).

- Randomly partition the data into  $k$  mutually exclusive subsets (folds), each approximately equal size.
- At  $i$ -th iteration, use  $D_i$  as test set and the others as training set.
- Average accuracy of all iterations.
- **Leave-one-out:**  $k$  folds, on  $i$ -th iteration leave out  $i$ -th fold; for small-sized data.
- **Stratified cross-validation:** For every class select a simple random sample of tuples. Results in subsets with approximately the same distribution.

**Example:**  $k$ -fold cross validation with  $k = 5$



## Evaluation Strategy: Bootstrap

**Bootstrap** samples training data uniformly with replacement.

**Several bootstrap methods exists, yet a common one is .632 bootstrap.**

- Data set with  $d$  tuples is sampled  $d$  times - uniformly with replacement.
- Uniformly = every tuple has the same probability ( $\frac{1}{d}$ ) for selection.
- With replacement = High chance a tuple is selected more than once.
- Not selected tuples will form the test set.
- Probability of not being chosen is  $1 - \frac{1}{d}$ . Selecting  $d$  times:  $(1 - \frac{1}{d})^d$ .
- With a large data set it approaches  $e^{-1} = 0.368$ .
- Thus, on average 63.2% of tuples are selected as the training set.
- Sampling procedure is repeated  $k$  times.

Calculate accuracy in every iteration as follows:

$$\text{Acc}(M) = \frac{1}{k} \sum_{i=1}^k 0.632 \cdot \text{Acc}(M_i)_{\text{test\_set}} + 0.368 \cdot \text{Acc}(M_i)_{\text{train\_set}}.$$

## Evaluating Classifier Accuracy: Bootstrap (II)

- **Suppose we have 2 classifiers,  $M_1$  and  $M_2$ , which one is better?**
- **Use 10-fold cross-validation to obtain  $\overline{\text{err}}(M_1)$  and  $\overline{\text{err}}(M_2)$ .**
  - Recall: error rate is  $1 - \text{accuracy}(M)$ .
- **Mean error rates:**
  - Just estimates of error on the true population of future data cases.
- **What if the difference between the 2 error rates is just attributed to chance?**
  - Use a test of statistical significance.
  - Obtain confidence limits for our error estimates.

## Evaluating Classifier Accuracy: Null Hypothesis

- **Perform  $k$ -fold cross-validation** with  $k = 10$ .
- **Assume samples follow a  $t$ -distribution** with  $k - 1$  degrees of freedom.
- **Use  $t$ -test**
  - Student's  $t$ -test.
- **Null hypothesis:**
  - $M_1$  and  $M_2$  are the same.
- **If we can reject the null hypothesis, then**
  - Conclude that difference between  $M_1$  and  $M_2$  is statistically significant.
  - Obtain confidence limits for our error estimates.

## Estimating Confidence Intervals: $t$ -Test

- **If only one test set available: pairwise comparison:**

- For  $i$ -th round of 10-fold cross-validation, the same cross partitioning is used to obtain  $\text{err}(M_1)_i$  and  $\text{err}(M_2)_i$ .
- Average over 10 rounds to get  $\overline{\text{err}}(M_1)$  and  $\overline{\text{err}}(M_2)$ .
- $t$ -test computes  $t$ -statistic with  $k - 1$  degrees of freedom:

$$t = \frac{\overline{\text{err}}(M_1) - \overline{\text{err}}(M_2)}{\frac{\text{var}(M_1 - M_2)}{\sqrt{k}}},$$

- where

$$\text{var}(M_1 - M_2) = \frac{1}{k} \sum_{i=1}^k [\text{err}(M_1)_i - \text{err}(M_2)_i - (\overline{\text{err}}(M_1) - \overline{\text{err}}(M_2))]^2.$$

- **If two test sets available: use unpaired  $t$ -test:**

$$\text{var}(M_1 - M_2) = \sqrt{\frac{\text{var}(M_1)}{k_1} + \frac{\text{var}(M_2)}{k_2}},$$

where  $k_1$  &  $k_2$  are # of cross-validation samples used for  $M_1$  &  $M_2$ , respectively.

# Estimating Confidence Intervals: Table for $t$ -Distribution

- Symmetrical.
- **Significance level:**
  - E.g.,  $\text{sig} = 0.05$  or  $5\%$  means  $M_1$  &  $M_2$  are significantly different for  $95\%$  of population.
- Confidence limit:  $z = \frac{\text{sig}}{2}$ .

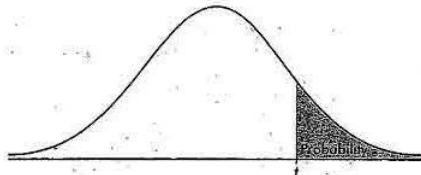


TABLE B:  $t$ -DISTRIBUTION CRITICAL VALUES

df	Tail probability $p$											
	.25	.20	.15	.10	.05	.025	.02	.01	.005	.0025	.001	.0005
1	1.000	1.376	1.963	3.078	6.314	12.71	15.89	31.82	63.66	127.3	318.3	636.6
2	.816	1.061	1.386	1.886	2.920	4.303	4.849	6.965	9.925	14.09	22.33	31.60
3	.765	.978	1.250	1.638	2.353	3.182	3.482	4.541	5.841	7.453	10.21	12.92
4	.741	.941	1.190	1.533	2.132	2.776	2.999	3.747	4.604	5.598	7.173	8.610
5	.727	.920	1.156	1.476	2.015	2.571	2.757	3.365	4.032	4.773	5.893	6.869
6	.718	.906	1.134	1.440	1.943	2.447	2.612	3.143	3.707	4.317	5.208	5.959
7	.711	.896	1.119	1.415	1.895	2.365	2.517	2.998	3.499	4.029	4.785	5.408
8	.706	.889	1.108	1.397	1.860	2.306	2.449	2.896	3.355	3.833	4.501	5.041
9	.703	.883	1.100	1.383	1.833	2.262	2.398	2.821	3.250	3.690	4.297	4.781
10	.700	.879	1.093	1.372	1.812	2.228	2.359	2.764	3.169	3.581	4.144	4.587
11	.697	.876	1.088	1.363	1.796	2.201	2.328	2.718	3.106	3.497	4.025	4.437
12	.695	.873	1.083	1.356	1.782	2.179	2.303	2.681	3.055	3.428	3.930	4.318
13	.694	.870	1.079	1.350	1.771	2.160	2.282	2.650	3.012	3.372	3.852	4.221
14	.692	.868	1.076	1.345	1.761	2.145	2.264	2.624	2.977	3.326	3.787	4.140
15	.691	.866	1.074	1.341	1.753	2.131	2.249	2.602	2.947	3.286	3.733	4.073
16	.690	.865	1.071	1.337	1.746	2.120	2.235	2.583	2.921	3.252	3.686	4.015
17	.689	.863	1.069	1.333	1.740	2.110	2.224	2.567	2.898	3.222	3.646	3.965
18	.688	.862	1.067	1.330	1.734	2.101	2.214	2.552	2.878	3.197	3.611	3.922
19	.688	.861	1.066	1.328	1.729	2.093	2.205	2.539	2.861	3.174	3.579	3.883
20	.687	.860	1.064	1.325	1.725	2.086	2.197	2.528	2.845	3.153	3.552	3.850
21	.686	.859	1.063	1.323	1.721	2.080	2.189	2.518	2.831	3.135	3.527	3.819
22	.686	.858	1.061	1.321	1.717	2.074	2.183	2.508	2.819	3.119	3.505	3.792
23	.685	.858	1.060	1.319	1.714	2.069	2.177	2.500	2.807	3.104	3.485	3.768
24	.685	.857	1.059	1.318	1.711	2.064	2.172	2.492	2.797	3.091	3.467	3.745
25	.684	.856	1.058	1.316	1.708	2.060	2.167	2.485	2.787	3.078	3.450	3.725
26	.684	.856	1.058	1.315	1.706	2.056	2.162	2.479	2.779	3.067	3.435	3.707
27	.684	.855	1.057	1.314	1.703	2.052	2.158	2.473	2.771	3.057	3.421	3.690
28	.683	.855	1.056	1.313	1.701	2.048	2.154	2.467	2.763	3.047	3.408	3.674
29	.683	.854	1.055	1.311	1.699	2.045	2.150	2.462	2.756	3.038	3.396	3.659
30	.683	.854	1.055	1.310	1.697	2.042	2.147	2.457	2.750	3.030	3.385	3.646
40	.681	.851	1.050	1.303	1.684	2.021	2.123	2.423	2.704	2.971	3.307	3.551
50	.679	.849	1.047	1.299	1.676	2.009	2.109	2.403	2.678	2.937	3.261	3.496
60	.679	.848	1.045	1.296	1.671	2.000	2.099	2.390	2.660	2.915	3.232	3.460
80	.678	.846	1.043	1.292	1.664	1.990	2.088	2.374	2.639	2.887	3.195	3.416
100	.677	.845	1.042	1.290	1.660	1.984	2.081	2.364	2.626	2.871	3.174	3.390
1000	.675	.842	1.037	1.282	1.646	1.962	2.056	2.330	2.581	2.813	3.098	3.300
$\infty$	.674	.841	1.036	1.282	1.645	1.960	2.054	2.326	2.576	2.807	3.091	3.291
Confidence level $C$												
	50%	60%	70%	80%	90%	95%	96%	98%	99%	99.5%	99.8%	99.9%



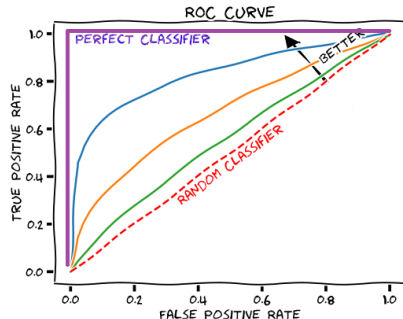
## Estimating Confidence Intervals: Statistical Significance

### Are $M_1$ and $M_2$ significantly different?

- Compute  $t$ . Select significance level (E.g.,  $\text{sig} = 5\%$ ).
- Consult table for  $t$ -distribution:
  - $t$ -distribution is symmetrical:
    - Typically upper % points of distribution shown.
  - Find critical value  $c$  corresponding to  $k - 1$  degrees of freedom (here, 9)
  - and for confidence limit  $z = \frac{\text{sig}}{2}$  (here, 0.025).
  - $\implies$  Here, critical value  $c = 2.262$
- If  $t > c$  or  $t < -c$ , then  $t$  value lies in rejection region:
  - **Reject null hypothesis** that mean error rates of  $M_1$  and  $M_2$  are equal.
  - Conclude: **statistically significant difference** between  $M_1$  and  $M_2$ .
- Otherwise, conclude that any difference is chance.

# Model Selection: Receiver Operating Characteristics (ROC) Curves

- Visual comparison of classification models.
- Compares and shows *trade-off* between TPR and FPR:
  - True Positive Rate (**TPR**): Proportion of positive tuples correctly classified as positive.  
→ sensitivity or recall =  $\frac{TP}{P}$
  - False Positive Rate (**FPR**): Proportion of negative tuples correctly classified as negative.  
→  $FPR = \frac{FP}{N} = 1 - \text{Specificity}$
- The area under the ROC curve is a **measure of the accuracy** of the model. Maximum area of 1.0 for a perfect classifier.
- The closer to the diagonal line (i.e. the closer the area is to 0.5), the less accurate is the model.



How to draw: Order tuples in decreasing order of probability.

- If TP move up TPR and plot point.
- If negative tuple classified as positive: move both FPR and FPR.

## Other Aspects of Model Selection

- **Speed**
  - Computational cost to train a classifier.
  - Time to use model (prediction time).
- **Robustness**, i. e. the ability to make accurate predictions.
  - Noisy data.
  - Missing values.
- **Scalability**, i. e. efficient construction of classifiers on an abundant amount of training tuples.
- **Interpretability**, refers to understanding and insight
  - Typically subjective and difficult to access.
  - Decision trees and classification rules are easy to interpret, but interpretability degrades with the size.
- *Other measures* such as goodness of rules, decision-tree size or compactness of classification rules.

---

# Ensemble Methods: Increasing the Accuracy

# Ensemble Methods

## Ensemble Method

An *ensemble method* creates a composite model that consists of several models such as to form one model.

- Most of the time *weak* learners are combined to mitigate their respective individual shortcomings.
- Data set is partitioned into  $k$  training sets.
- Train a classifier on each training set.
- Every individual classifier returns its prediction.
- Overall prediction is determined for instance by *majority voting*.
- Prediction typically more accurate than each individual model.  
→ Returns better results when diversity among models is great.
- *Each classifier should perform better than random guessing*.
- **Popular methods** include Bagging, Boosting, and Random Forests.

## Bagging: Bootstrap Aggregation

- **Analogy:**
  - Diagnosis based on multiple doctors' majority vote.
- **Training:**
  - Given a set  $D$  of  $d$  tuples, at each iteration  $i$ , a training set  $D_i$  of  $d$  tuples is sampled with replacement from  $D$  (i.e., bootstrap).
  - A classifier model  $M_i$  is learned for each training set  $D_i$ .
- **Prediction in the case of classification: classify an unknown sample  $X$ .**
  - Each classifier  $M_i$  returns its class prediction.
  - The bagged classifier  $M^*$  counts the votes and assigns the class with the most votes to  $X$ .
- **Prediction of a real number** (i. e. regression or time series forecast):
  - Can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple.
- **Accuracy:**
  - Often significantly better than a single classifier derived from  $D$ .
  - For noisy data: not considerably worse, more robust.
  - Proved improved accuracy in prediction.

# Boosting

- **Analogy:**
  - Consult several doctors, based on a combination of weighted diagnoses – weight assigned based on the previous diagnosis accuracy
- **How boosting works:**
  - Weights are assigned to each training tuple.
  - A series of  $k$  classifiers is iteratively learned.
  - After a classifier  $M_i$  is learned, the weights are updated to allow the subsequent classifier,  $M_{i+1}$  to pay more attention to the training tuples that were misclassified by  $M_i$ .
  - The final  $M^*$  combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy.
- **Classification:**
  - Each classifier  $M_i$  returns its class prediction.
  - The bagged classifier  $M^*$  counts the votes and assigns the class with the most votes to  $X$ .
- **Boosting algorithm can be extended for numeric prediction.**

## AdaBoost ("Adaptive Boosting"<sup>2</sup>): Training

- **Given a data set  $D$  of  $d$  class-labeled tuples:**  $(x_1, y_1), \dots, (x_d, y_d)$  with  $y_d \in Y = \{1, \dots, c\}$ .
- **Initialize empty lists to hold information per classifier:**  $\mathbf{w}, \beta, \mathbf{M} \leftarrow$  empty list.
- **Initialize weights for first classifier to hold same probability for each tuple:**  $w_j^1 \leftarrow \frac{1}{d}$
- **Generate  $K$  classifiers in  $K$  iterations. At iteration  $k$ ,**
  1. Calculate "normalized" weights:  $\mathbf{p}^k = \frac{\mathbf{w}^k}{\sum_{j=1}^d w_j^k}$
  2. Sample dataset with replacement according to  $\mathbf{p}^k$  to form training set  $D_k$ .
  3. Derive classification model  $M_k$  from  $D_k$ .
  4. Calculate error  $\varepsilon_k$  by using  $D_k$  as a test set as follows:  $\varepsilon_k = \sum_{j=1}^d p_j^k \cdot \text{err}(M_k, x_j, y_j)$ ,  
where the *misclassification error*  $\text{err}(M_k, x_j, y_j)$  returns 1 if  $M_k(x_j) \neq y_j$ , otherwise it returns 0.
  5. If  $\text{error}(M_k) > 0.5$ : Abandon this classifier and go back to step 1.
  6. Calculate  $\beta_k = \frac{\varepsilon_k}{1 - \varepsilon_k}$ .
  7. Update weights for the next iteration:  $w_j^{k+1} = w_j^k \beta_k^{1 - \text{err}(M_k, x_j, y_j)}$ . *If a tuple is misclassified, its weight remains the same, otherwise it is decreased.* Misclassified tuple weights are increased relatively.
  8. Add  $\mathbf{w}^{k+1}$ ,  $M_k$ , and  $\beta_k$  to their respective lists.

<sup>2</sup>Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997. DOI: 10.1006/jcss.1997.1504. [Online]. Available: <https://doi.org/10.1006/jcss.1997.1504>, Algorithm AdaBoost.M1 on p. 131.



## AdaBoost ("Adaptive Boosting"<sup>3</sup>): Prediction

- **Initialize weight of each class to zero.**
- **For each classifier  $i$  in  $k$  classifiers:**
  1. Calculate the weight of this classifier's vote:  $w_i = \log(\frac{1}{\beta_i})$ .
  2. Get class prediction  $c$  for (single) tuple  $x$  from current weak classifier  $M_i$ :  $c = M_i(x)$ .
  3. Add  $w_i$  to weight for class  $c$ .
- **Return predicted class with the largest weight.**
- Mathematically, this can be formulated as:

$$M(x) = \arg \max_{y \in Y} \sum_{i=1}^k (\log \frac{1}{\beta_i}) M_i(x)$$

<sup>3</sup>Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997. DOI: 10.1006/jcss.1997.1504. [Online]. Available: <https://doi.org/10.1006/jcss.1997.1504>, Algorithm AdaBoost.M1 on p. 131.

## Random Forest<sup>4</sup>

- Ensemble method consisting only of decision trees where each tree has been generated using random selection of attributes at each node.
- Classification: Each tree votes and the most popular class is returned.
- **Two methods to construct random forests:** (each builds  $k$  trees)
  1. Forest-RI (random input selection):
    - Random sampling with replacement to obtain training data from  $D$ .
    - Set  $F$  as the number of attributes to determine split at each node.  $F$  is smaller than the number of available attributes.
    - Construct decision tree  $M_i$  by randomly select candidates at each node. Use CART to grow tree to maximum size without pruning.
  2. Forest-RC: Similar to Forest-RI but new attributes (features) are generated by linear combinations of existing attributes to reduce correlation between individual classifiers. At each node, attributes are randomly selected.
- **Comparable in accuracy to AdaBoost, but more robust to errors and outliers.**
- **Insensitive to the number of attributes selected for consideration at each split, and faster than bagging or boosting.**

<sup>4</sup>L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001. DOI: 10.1023/A:1010933404324. [Online]. Available: <https://doi.org/10.1023/A:1010933404324>.

# Classification of Class-imbalanced Data Sets

## Class-Imbalanced Data

*Class-Imbalanced Data* refers to data where the main class of interest (positive labeled) is only represented by a small number of tuples. E.g., medical diagnosis and fraud detection.

- Problem because traditional methods assume *equality between classes*, i. e. a balanced distribution of classes and equal error costs.
- **Typical methods for imbalanced data in binary classification:**
  1. **Undersampling/Oversampling:** Changes distribution of tuples in training data.
    - *Undersampling:* Randomly eliminate tuples from negative class.
    - *Oversampling:* Re-samples data from positive class.  
For instance, method SMOTE generates synthetic data that is similar to existing data using nearest neighbor.
  2. **Threshold-moving:** Moves the decision threshold,  $t$ , so that the rare-class tuples are easier to classify, and hence, less chance of costly false-negative errors. Works when class returns a probability.
  3. **Ensemble techniques.**

Threshold-moving and ensemble methods work well on extremely imbalanced data.

- **Still difficult on multi-class tasks.**

---

# Summary


## Summary

- **Classification.**
  - A form of data analysis that extracts models describing important data classes.
- **Effective and scalable methods developed for:**
  - Decision-tree induction, naive Bayesian classification, rule-based classification, and many other classification methods.
- **Evaluation metrics:**
  - Accuracy, sensitivity, specificity, precision, recall,  $F$ -measure, and  $F_{\beta}$ -measure.
- **Stratified  $k$ -fold cross-validation.**
  - Recommended for accuracy estimation.
- **Significance tests and ROC curves.**
  - Useful for model selection.
- **Ensemble methods and class-imbalanced data**
  - Boosting, Bagging (AdaBoost), and Random Forests.
  - Methods to mitigate class-imbalanced problem for binary classification.

## Any questions about this chapter?

Ask them now or ask them later in our forum:

StudOn Forum

 <https://www.studon.fau.de/frm5699567.html>

---

# Appendix

## Data:

- Training dataset  $D$  containing tuples with their associated class labels;
- `attribute_list`, the set of candidate attributes;
- `attribute_selection_method`, a method to determine the splitting criterion that “best” partitions the data tuples into individual classes. The criterion consists of a `splitting_attribute`, and possibly, either a `split_point` or `splitting_subset`.

## Result: A decision tree.

```
1 create a node  $N$ ;  
2 if tuples in  $D$  are all of the same class  $C$  then  
3   | return  $N$  as a leaf node labeled with the class  $C$ ;  
4 if attribute_list is empty then  
5   | /* Majority voting  
6   |   majority_class  $\leftarrow$  determine majority class in  $D$ ;  
   | return  $N$  as a leaf node labeled with majority_class;
```

```
/* apply attribute_selection_method to find the  
   “best” splitting_criterion
```

```
7   splitting_criterion  $\leftarrow$  attribute_selection_method( $D$ ,  
   attribute_list);  
8   label node  $N$  with splitting_criterion;  
9   if (splitting_attribute is discrete-valued and multiway splits  
       allowed) or attribute value has only one unique value then  
   |   /* remove splitting_attribute  
10  |   attribute_list  $\leftarrow$  attribute_list - splitting_attribute;  
11  foreach outcome  $j$  of splitting_criterion do  
   |   /* partition the tuples and grow subtrees for  
   |   each partition */  
12  |    $D_j \leftarrow$  partition  $D$  to satisfy outcome  $j$ ;  
13  |   if  $D_j$  is empty then  
14  |   | attach a leaf labeled with the majority class in  $D$  to node  $N$ ;  
15  |   else  
16  |   | attach the node return by build_decision_tree( $D_j$ ,  
   |   attribute_list) to node  $N$ ;  
17  return  $N$ ;  
*/
```

**Algorithm 1:** `build_decision_tree`. Generate a decision tree from training tuples in data partition  $D$ .