# Classification and regression trees

Pierre Geurts
p.geurts@ulg.ac.be

Last update: 23/09/2015

# Outline

- Supervised learning
- Decision tree representation
- Decision tree learning
- Extensions
- Regression trees
- By-products

# Database

- A collection of objects (rows) described by attributes (columns)

| checkingaccount | duration | purpose | amount | savings | yearsemployed | age | good or bad |
|---|---|---|---|---|---|---|---|
| 0<=…<200 DM | 48 | radiotv | 5951 | ...<100 DM | 1<...<4 | 22 | bad |
| ...<0 DM | 6 | radiotv | 1169 | unknown | ...>7 | 67 | good |
| no | 12 | education | 2096 | ...<100 DM | 4<...<7 | 49 | good |
| ...<0 DM | 42 | furniture | 7882 | ...<100 DM | 4<...<7 | 45 | good |
| ...<0 DM | 24 | newcar | 4870 | ...<100 DM | 1<...<4 | 53 | bad |
| no | 36 | education | 9055 | unknown | 1<...<4 | 35 | good |
| no | 24 | furniture | 2835 | 500<...<1000 DM | ...>7 | 53 | good |
| 0<=...<200 DM | 36 | usedcar | 6948 | ...<100 DM | 1<...<4 | 35 | good |
| no | 12 | radiotv | 3059 | ...>1000 DM | 4<...<7 | 61 | good |
| 0<=...<200 DM | 30 | newcar | 5234 | ...<100 DM | unemployed | 28 | bad |
| 0<=...<200 DM | 12 | newcar | 1295 | ...<100 DM | ...<1 | 25 | bad |
| ...<0 DM | 48 | business | 4308 | ...<100 DM | ...<1 | 24 | bad |
| 0<=...<200 DM | 12 | radiotv | 1567 | ...<100 DM | 1<...<4 | 22 | good |

# Supervised learning

inputs output

| $A_1$ | $A_2$ | … | $A_n$ | $Y$ |
|---|---|---|---|---|
| 2.3 | on | | 3.4 | C1 |
| 1.2 | off | | 0.3 | C2 |
| … | … | | … | … |

Database=learning sample

Automatic learning →

$$\hat{Y} = f(A_1, A_2, \ldots, A_n)$$

model

- Goal: from the database, find a function f of the inputs that approximate at best the output

- Discrete output → classification problem
- Continuous output → regression problem

# Examples of application (1)

- Predict whether a bank client will be a good debtor or not
- Image classification:
  - Handwritten characters recognition:

 → 3  → 5

  - Face recognition

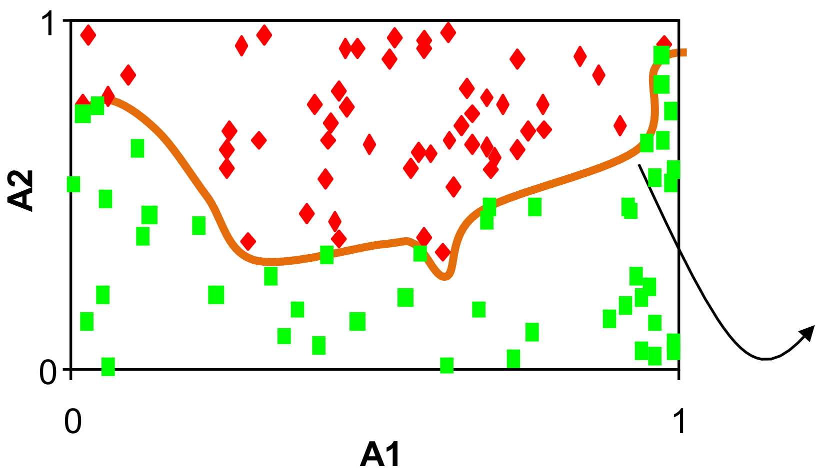
# Examples of application (2)

- Classification of cancer types from gene expression profiles (Golub et al (1999))

| N° patient | Gene 1 | Gene 2 | ... | Gene 7129 | Leucimia |
|---|---|---|---|---|---|
| 1 | -134 | 28 | ... | 123 | AML |
| 2 | -123 | 0 | ... | 17 | AML |
| 3 | 56 | -123 | ... | -23 | ALL |
| ... | ... | ... | ... | ... | ... |
| 72 | 89 | -123 | ... | 12 | ALL |

# Learning algorithm

- It receives a learning sample and returns a function $h$
- A learning algorithm is defined by:
  - A hypothesis space $H$ (=a family of candidate models)
  - A quality measure for a model
  - An optimisation strategy



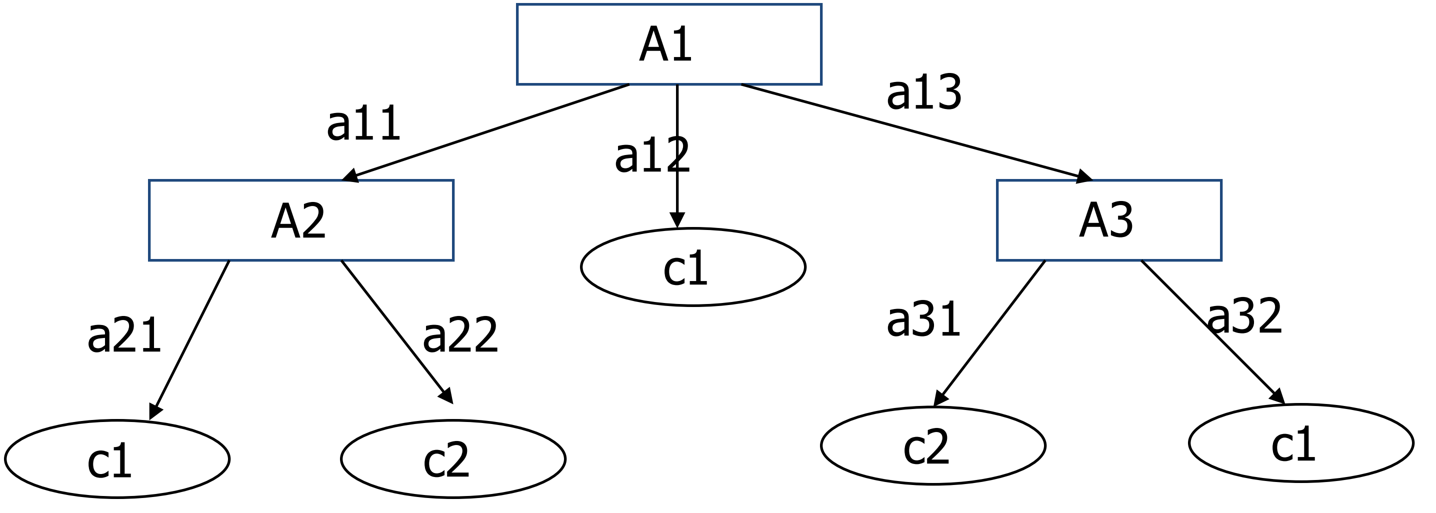

A model ($h \in H$) obtained by automatic learning

# Decision (classification) trees

- A learning algorithm that can handle:
  - Classification problems (binary or multi-valued)
  - Attributes may be discrete (binary or multi-valued) or continuous.
- Classification trees were invented twice:
  - By statisticians: CART (Breiman et al.)
  - By the AI community: ID3, C4.5 (Quinlan et al.)

# Hypothesis space

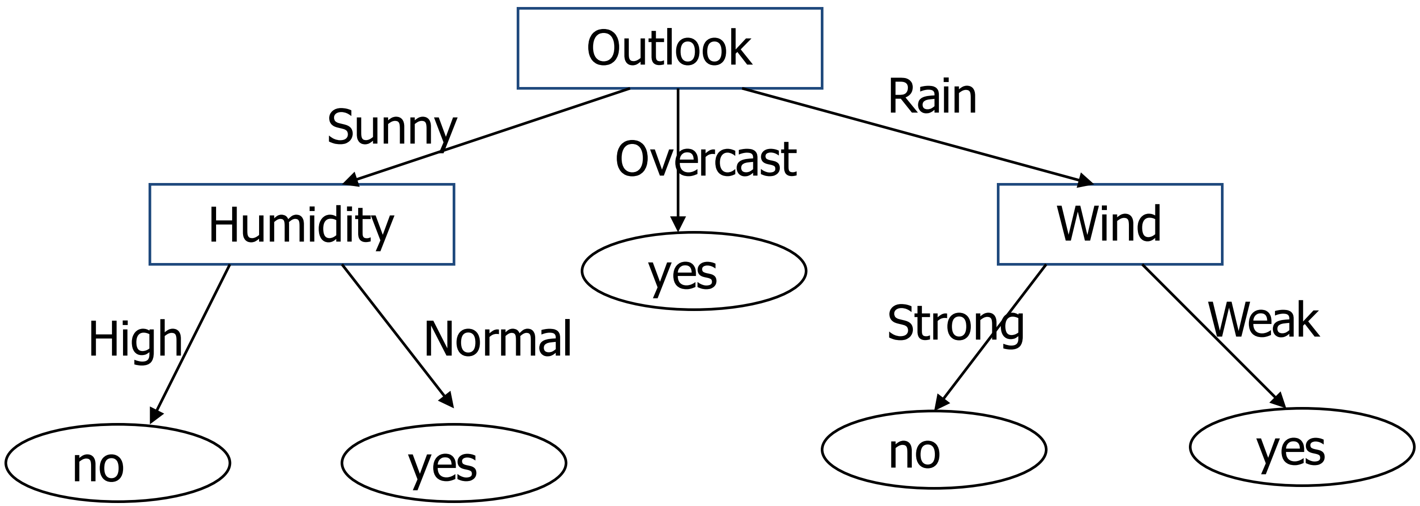
- A decision tree is a tree where:
  - Each *interior node* tests an attribute
  - Each *branch* corresponds to an attribute value
  - Each *leaf* node is labelled with a class



# A simple database: playtennis

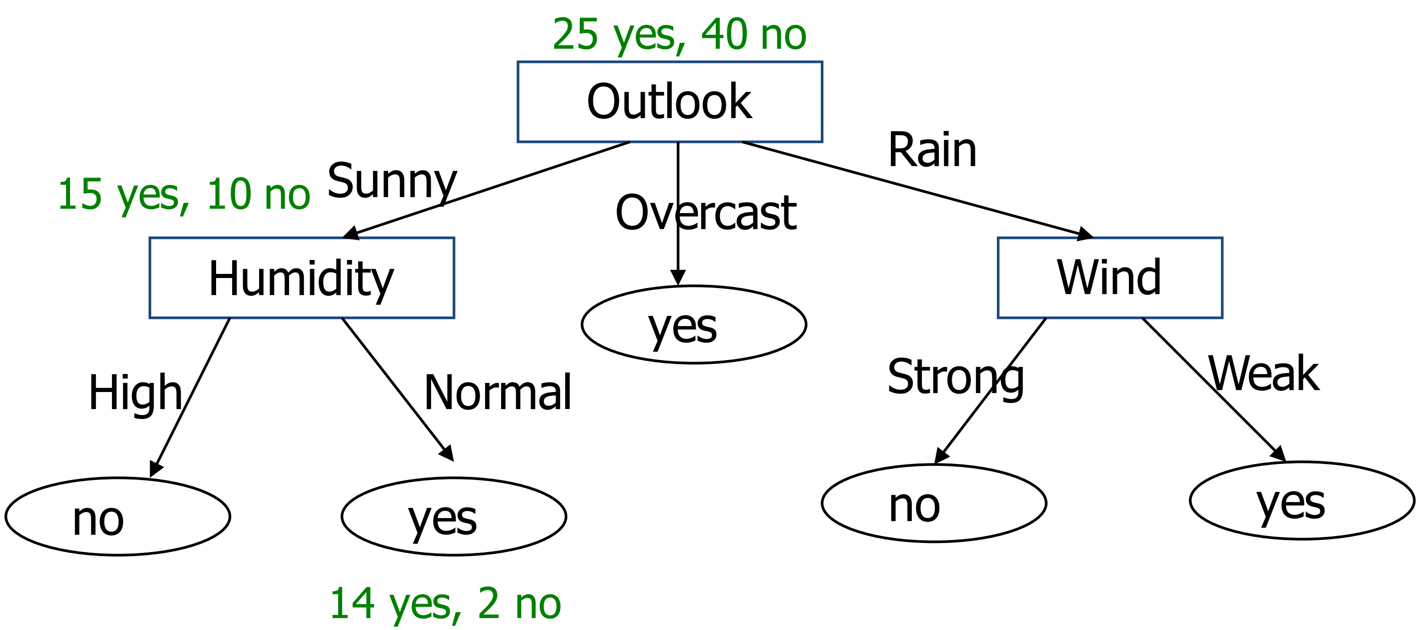
| Day | Outlook | Temperature | Humidity | Wind | Play Tennis |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | Normal | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | High | Strong | Yes |
| D8 | Sunny | Mild | Normal | Weak | No |
| D9 | Sunny | Hot | Normal | Weak | Yes |
| D10 | Rain | Mild | Normal | Strong | Yes |
| D11 | Sunny | Cool | Normal | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |
| D14 | Rain | Mild | High | Strong | No |

# A decision tree for playtennis



# Tree learning

- Tree learning=choose the tree structure and determine the predictions at leaf nodes
- Predictions: to minimize the misclassification error, associate the majority class among the learning sample cases reaching this node



# How to generate trees ? (1)

- What properties do we want the decision tree to have ?

1. It should be consistent with the learning sample (for the moment)
    - Trivial algorithm: construct a decision tree that has one path to a leaf for each example
    - Problem: it does not capture useful information from the database

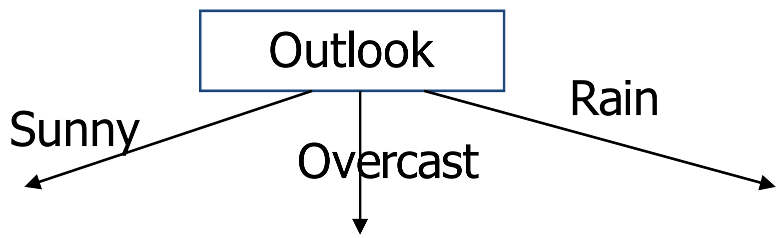
# How to generate trees ? (2)

- What properties do we want the decision tree to have ?

2. It should be at the same time as simple as possible
   - Trivial algorithm: generate all trees and pick the simplest one that is consistent with the learning sample.
   - Problem: intractable, there are too many trees

# Top-down induction of DTs (1)

- Choose « best » attribute
- Split the learning sample
- Proceed recursively until each object is correctly classified




| Day | Outlook | Temp. | Humidity | Wind | Play |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D8 | Sunny | Mild | High | Weak | [illegible] |
| D9 | Sunny | Hot | Normal | Weak | [illegible] |
| D11 | Sunny | Cool | Normal | Strong | [illegible] |

| Day | Outlook | Temp. | Humidity | Wind | Play |
|---|---|---|---|---|---|
| D3 | Overcast | Hot | High | Weak | Yes |
| D7 | Overcast | Cool | High | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |

| Day | Outlook | Temp. | Humidity | Wind | Play |
|---|---|---|---|---|---|
| D4 | Rain | Mild | Normal | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| [illegible] | Rain | Cool | Normal | Strong | No |
| [illegible] | Rain | Mild | Normal | Strong | Yes |
| [illegible] | Rain | Mild | High | Strong | No |

# Top-down induction of DTs (2)

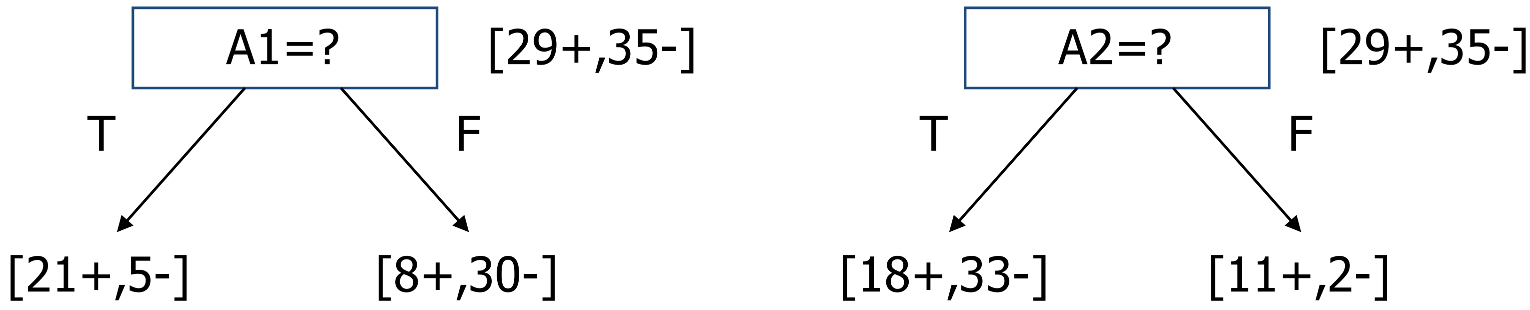
Procedure learn_dt(learning sample, $LS$)

- If all objects from $LS$ have the same class
  - Create a leaf with that class
- Else
  - Find the « best » splitting attribute $A$
  - Create a test node for this attribute
  - For each value $a$ of $A$
    - Build $LS_a = \{o \in LS \mid A(o) \text{ is } a\}$
    - Use Learn_dt($LS_a$) to grow a subtree from $LS_a$.

# Properties of TDIDT

- Hill-climbing algorithm in the space of possible decision trees.
  - It adds a sub-tree to the current tree and continues its search
  - It does not backtrack
- Sub-optimal but very fast
- Highly dependent upon the criterion for selecting attributes to test

# Which attribute is best ?




- We want a small tree
  - We should maximize the class separation at each step, i.e. make successors as pure as possible
  - $\Rightarrow$ it will favour short paths in the trees

# Impurity

- Let $LS$ be a sample of objects, $p_j$ the proportions of objects of class $j$ ($j$=1,…,$J$) in $LS$,
- Define an impurity measure $I(LS)$ that satisfies:
  - $I(LS)$ is minimum only when $p_i=1$ and $p_j=0$ for $j \neq i$ (all objects are of the same class)
  - $I(LS)$ is maximum only when $p_j = 1/J$ (there is exactly the same number of objects of all classes)
  - $I(LS)$ is symmetric with respect to $p_1, \ldots, p_J$

# Reduction of impurity

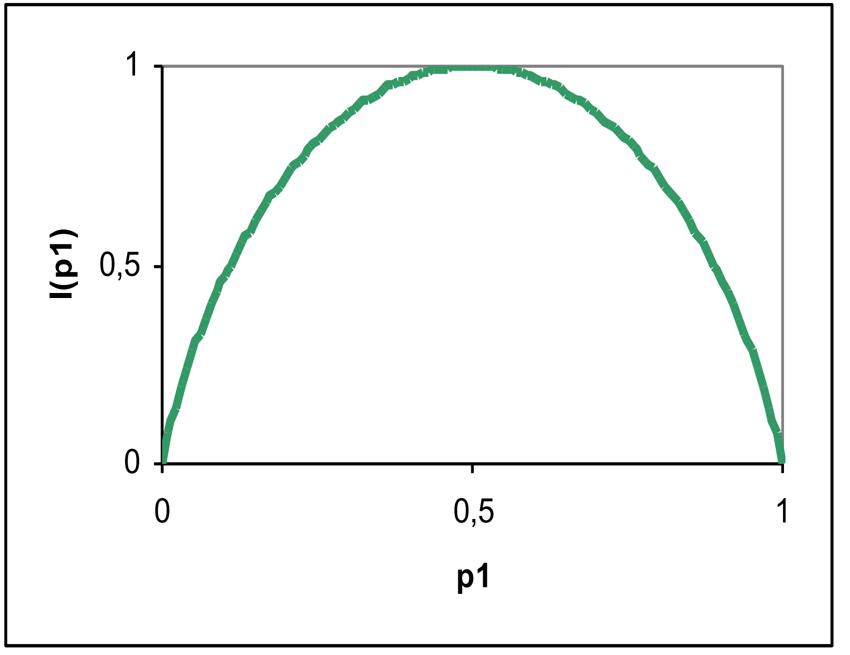
- The “best” split is the split that maximizes the expected reduction of impurity

$$\Delta I(LS, A) = I(LS) - \sum_a \frac{|LS_a|}{|LS|} I(LS_a)$$

  where $LS_a$ is the subset of objects from *LS* such that $A=a$.
- $\Delta I$ is called a score measure or a splitting criterion
- There are many other ways to define a splitting criterion that do not rely on an impurity measure

# Example of impurity measure (1)

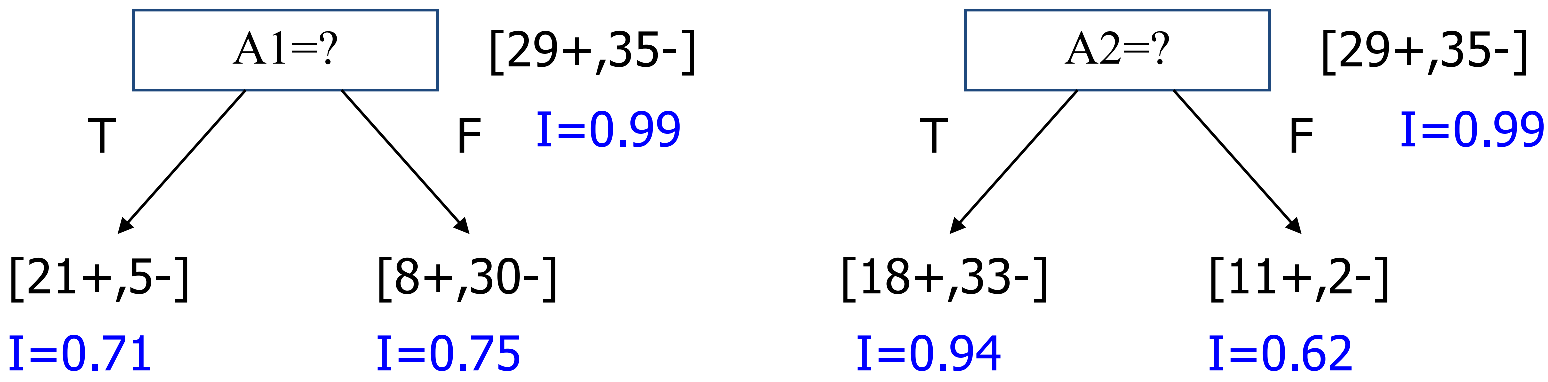
- Shannon's entropy:
  - $H(LS) = -\sum_j p_j \log p_j$
  - If two classes, $p_1 = 1 - p_2$




- Entropy measures impurity, uncertainty, surprise…
- The reduction of entropy is called the information gain

# Example of impurity measure (2)

- Which attribute is best ?




$$\Delta I(LS,A1) = 0.99 - (26/64)\ 0.71 - (38/64)\ 0.75$$

$$= 0.25$$

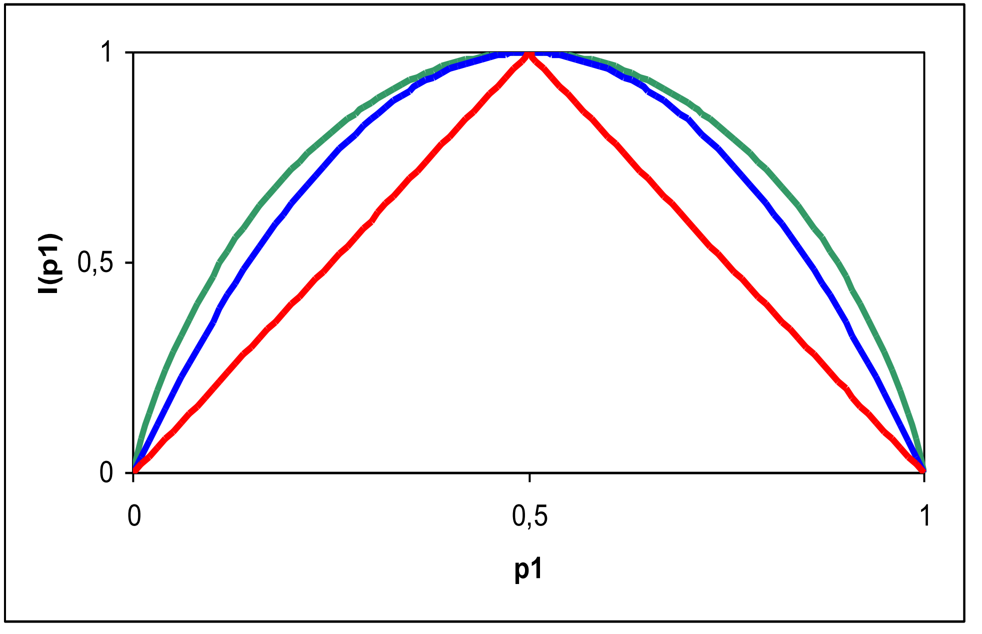
$$\Delta \mathrm{I}(LS,A2) = 0.99 - (51/64)\ 0.94 - (13/64)\ 0.62$$

$$= 0.12$$

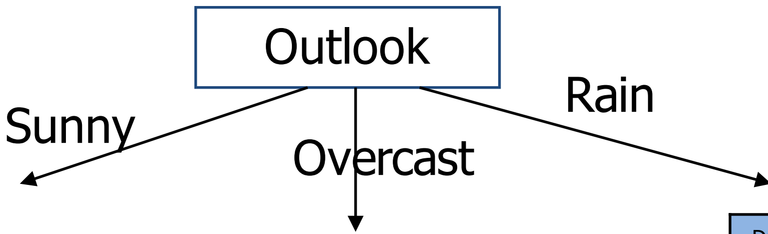
# Other impurity measures

- Gini index:
  - $I(LS)=\sum_j p_j\,(1\text{-}p_j)$
- Misclassification error rate:
  - $I(LS)=1\text{-}\max_j p_j$
- two-class case:

Green: entropy
Blue: Gini index
Red: misclas. error
(normalized between 0 and 1)



# Playtennis problem




| Day | Outlook | Temp. | Humidity | Wind | Play |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D8 | Sunny | Mild | High | Weak | No |
| D9 | Sunny | Hot | Normal | Weak | Yes |
| D11 | Sunny | Cool | Normal | Strong | Yes |

| Day | Outlook | Temp. | Humidity | Wind | Play |
|---|---|---|---|---|---|
| D3 | Overcast | Hot | High | Weak | Yes |
| D7 | Overcast | Cool | High | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |

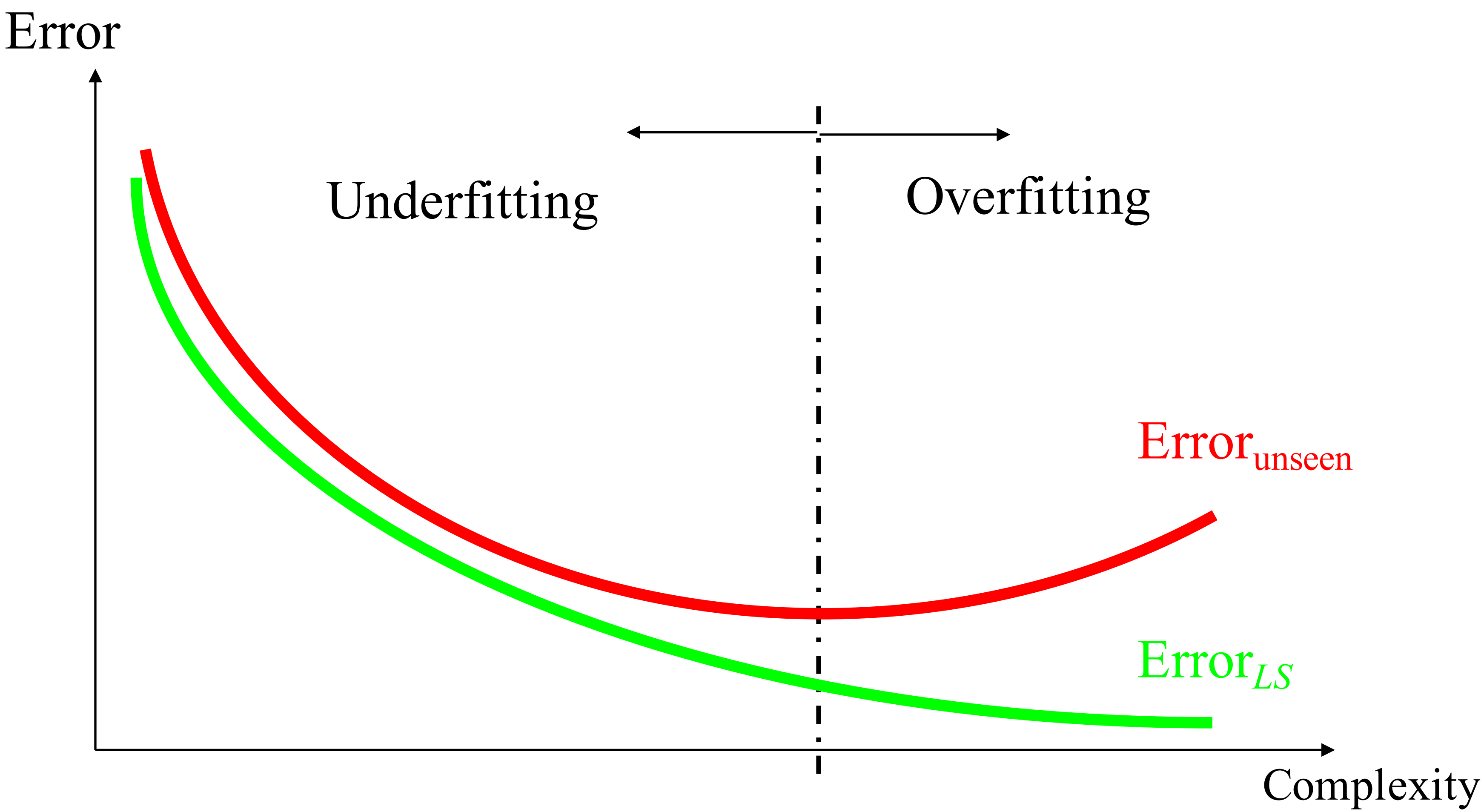
| Day | Outlook | Temp. | Humidity | Wind | Play |
|---|---|---|---|---|---|
| | Rain | Mild | Normal | Weak | Yes |
| | Rain | Cool | Normal | Weak | Yes |
| | Rain | Cool | Normal | Strong | No |
| | Rain | Mild | Normal | Strong | Yes |
| | Rain | Mild | High | Strong | No |

- Which attribute should be tested here ?
  - $\Delta I(LS,\text{Temp.}) = 0.970 - (3/5)\ 0.918 - (1/5)\ 0.0 - (1/5)\ 0.0 = 0.419$
  - $\Delta I(LS,\text{Hum.}) = 0.970 - (3/5)\ 0.0 - (2/5)\ 0.0 = 0.970$
  - $\Delta I(LS,\text{Wind}) = 0.970 - (2/5)\ 1.0 - (3/5)\ 0.918 = 0.019$
- $\Rightarrow$ the best attribute is Humidity

# Overfitting (1)

- Our trees are perfectly consistent with the learning sample
- But, often, we would like them to be good at predicting classes of unseen data from the same distribution (generalization).
- A tree T overfits the learning sample iff ∃ T' such that:
  - $Error_{LS}(T) < Error_{LS}(T')$
  - $Error_{unseen}(T) > Error_{unseen}(T')$

# Overfitting (2)

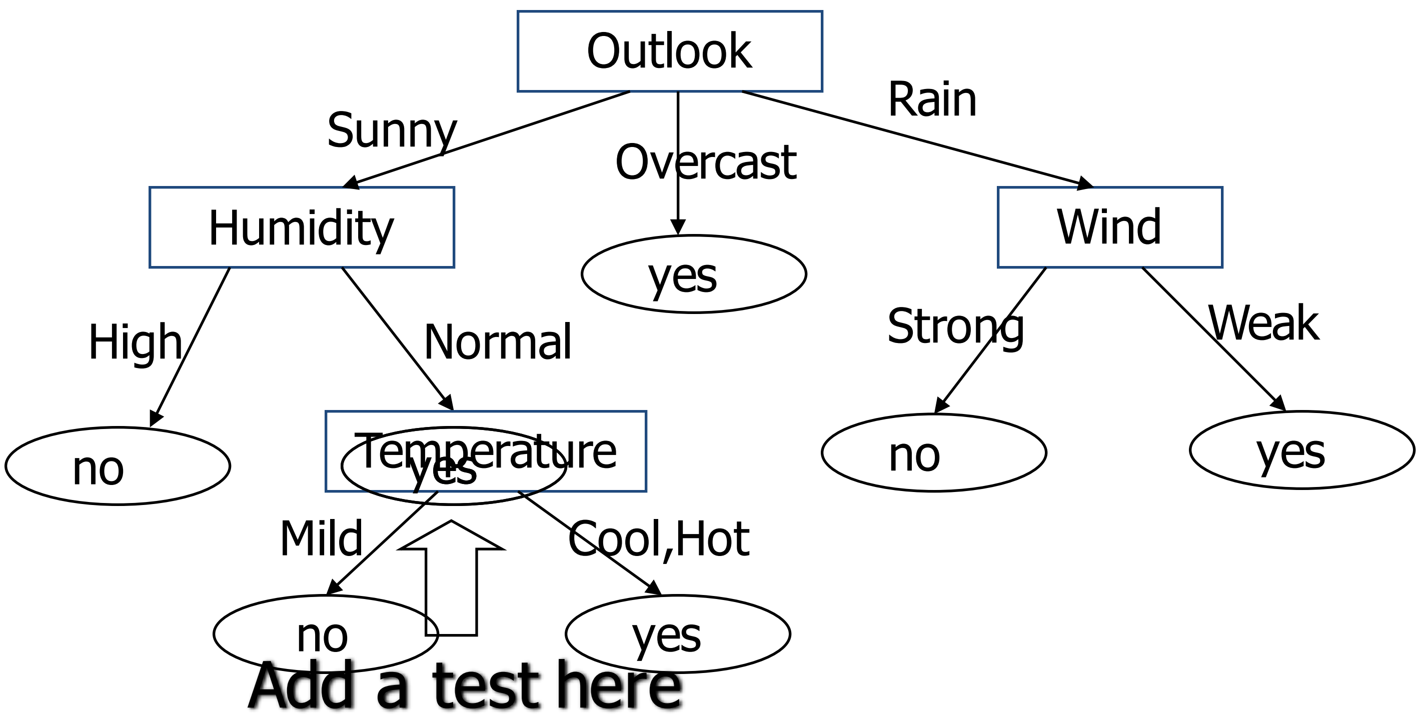



- In practice, $\text{Error}_{\text{unseen}}(T)$ is estimated from a separate test sample

# Reasons for overfitting (1)

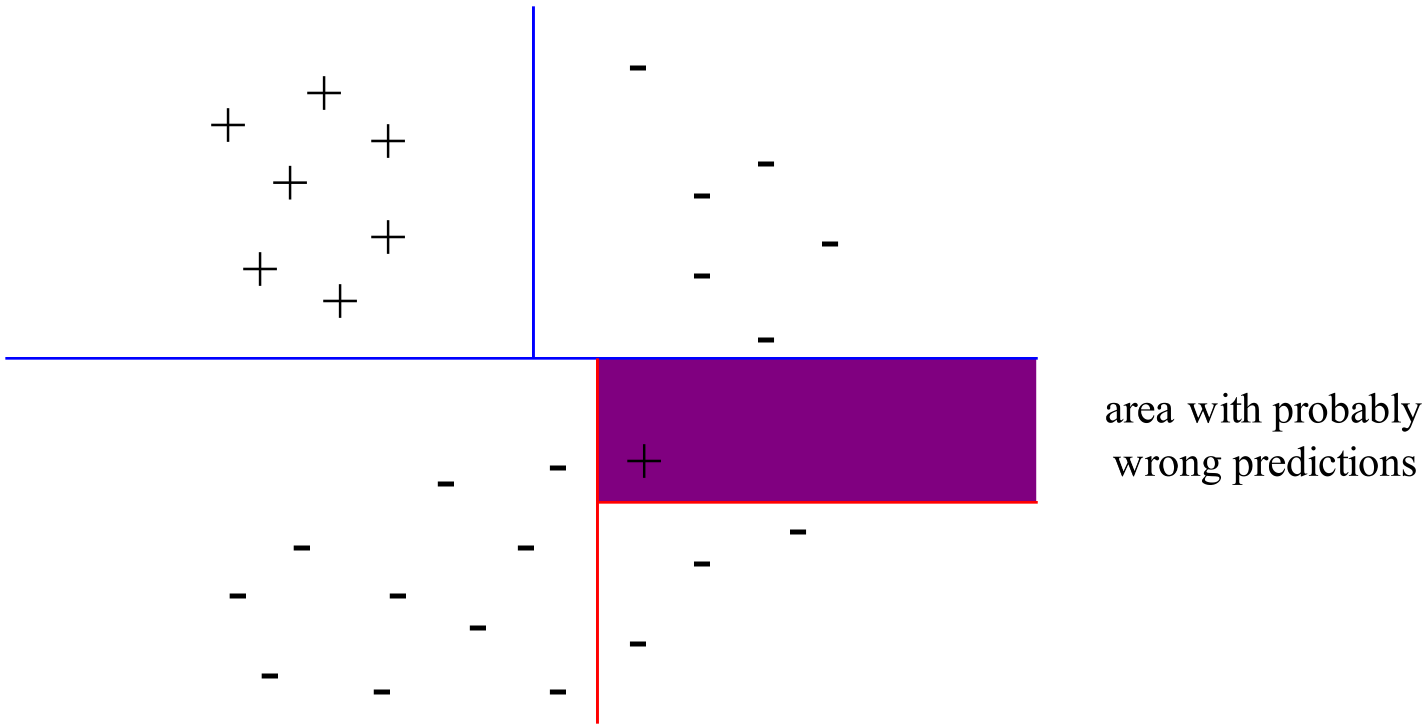
- Data is noisy or attributes do not completely predict the outcome

| Day | Outlook | Temperature | Humidity | Wind | Play Tennis |
|---|---|---|---|---|---|
| D15 | Sunny | Mild | Normal | Strong | No |



# Reasons for overfitting (2)

- Data is incomplete (not all cases covered)




- We do not have enough data in some part of the learning sample to make a good decision

# How can we avoid overfitting ?

- Pre-pruning: stop growing the tree earlier, before it reaches the point where it perfectly classifies the learning sample
- Post-pruning: allow the tree to overfit and then post-prune the tree
- Ensemble methods (later in this course)

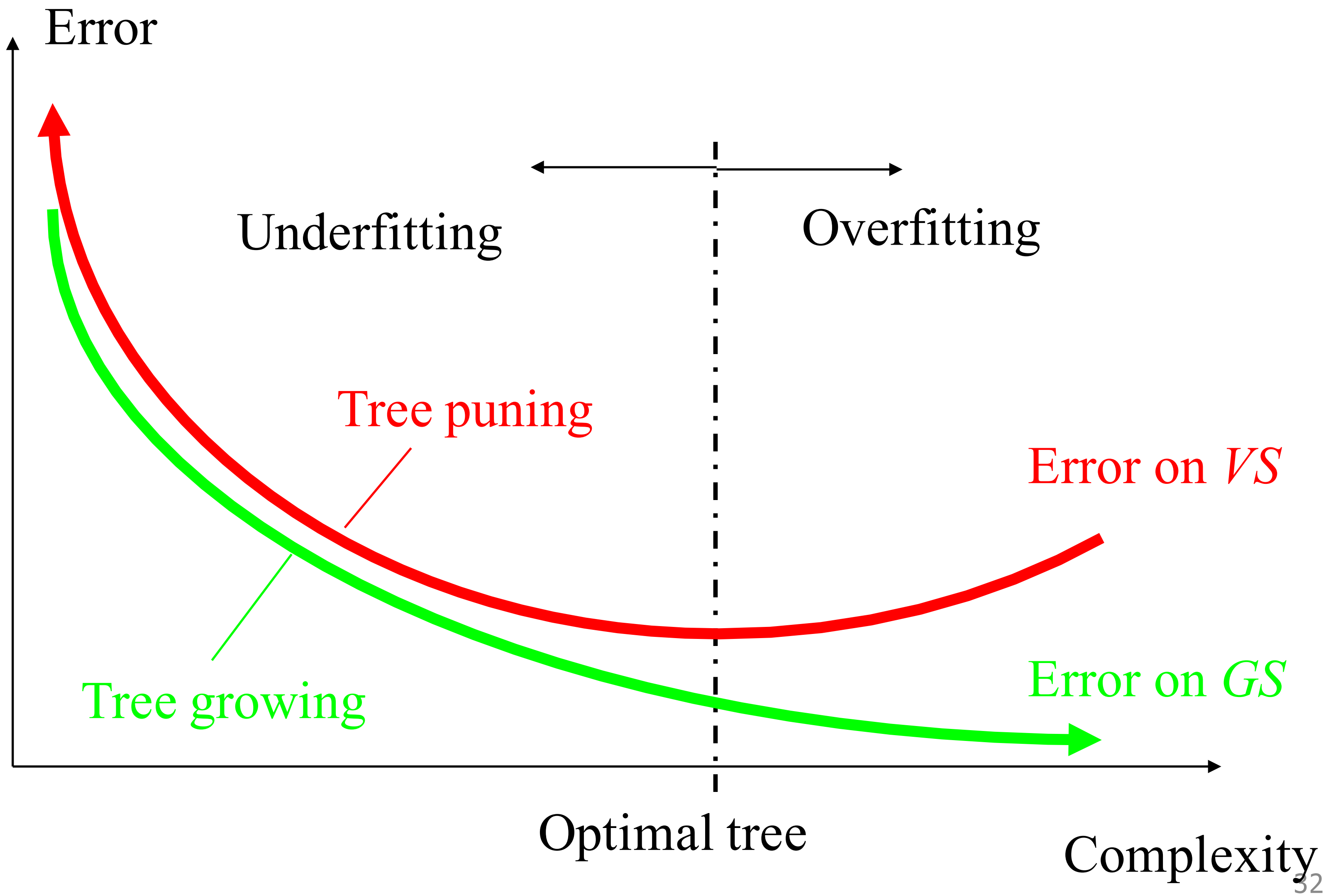
# Pre-pruning

- Stop splitting a node if
  - The number of objects is too small
  - The impurity is low enough
  - The best test is not statistically significant (according to some statistical test)
- Problem:
  - the optimum value of the parameter ($n$, $I_{th}$ , significance level) is problem dependent.
  - We may miss the optimum

# Post-pruning (1)

- Split the learning sample $LS$ into two sets:
  - a growing sample $GS$ to build the tree
  - A validation sample $VS$ to evaluate its generalization error
- Build a complete tree from $GS$
- Compute a sequence of trees $\{T_1,T_2,\ldots\}$ where
  - $T_1$ is the complete tree
  - $T_i$ is obtained by removing some test nodes from $T_{i-1}$
- Select the tree $T_i^*$ from the sequence that minimizes the error on $VS$

# Post-pruning (2)

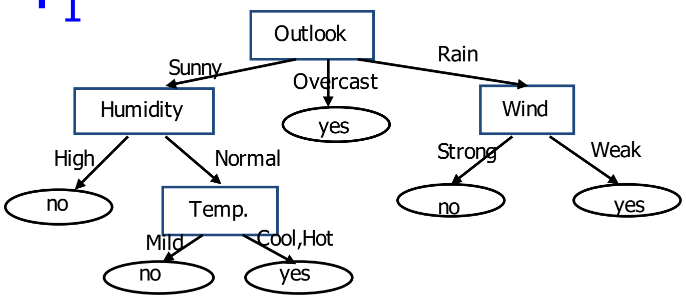


# Post-pruning (3)

- How to build the sequence of trees ?
  - Reduced error pruning:
    - At each step, remove the node that most decreases the error on $VS$
  - Cost-complexity pruning:
    - Define a cost-complexity criterion:
      - $\text{Error}_{GS}(\text{T})+\alpha.\text{Complexity}(\text{T})$
    - Build the sequence of trees that minimize this criterion for increasing $\alpha$

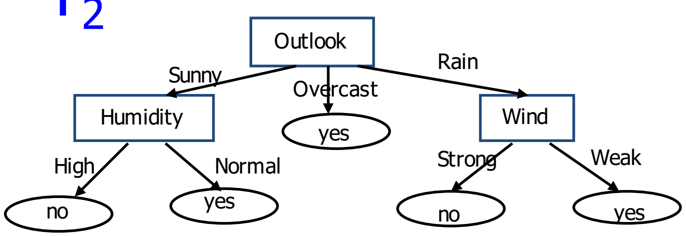
# Post-pruning (4)

$T_1$



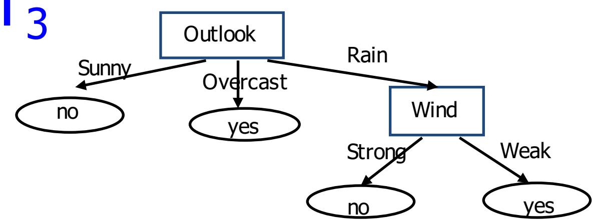

$Error_{GS}=0\%$, $Error_{VS}=10\%$

$T_2$



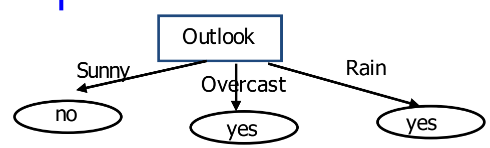

$Error_{GS}=6\%$, $Error_{VS}=8\%$

$T_3$



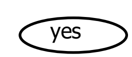

$Error_{GS}=13\%$, $Error_{VS}=15\%$

$T_4$




$Error_{GS}=27\%$, $Error_{VS}=25\%$

$T_5$




$Error_{GS}=33\%$, $Error_{VS}=35\%$

# Post-pruning (5)

- Problem: require to dedicate one part of the learning sample as a validation set ⇒ may be a problem in the case of a small database
- Solution: *N*-fold cross-validation
  - Split the training set into *N* parts (often 10)
  - Generate *N* trees, each leaving one part among *N*
  - Make a prediction for each learning object with the (only) tree built without this case.
  - Estimate the error of this prediction
- May be combined with pruning

# How to use decision trees ?

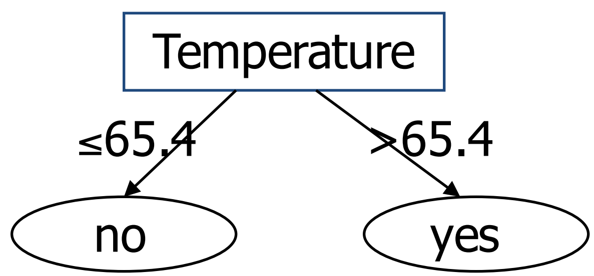
- Large datasets (ideal case):
  - Split the dataset into three parts: $GS$, $VS$, $TS$
  - Grow a tree from $GS$
  - Post-prune it from $VS$
  - Test it on $TS$
- Small datasets (often)
  - Grow a tree from the whole database
  - Pre-prune with default parameters (risky), post-prune it by 10-fold cross-validation (costly)
  - Estimate its accuracy by 10-fold cross-validation

# Outline

- Supervised learning
- Tree representation
- Tree learning
- Extensions
  - Continuous attributes
  - Attributes with many values
  - Missing values
- Regression trees
- By-products

# Continuous attributes (1)

- Example: temperature as a number instead of a discrete value
- Two solutions:
  - Pre-discretize: Cold if Temperature<70, Mild between 70 and 75, Hot if Temperature>75
  - Discretize during tree growing:




- How to find the cut-point ?

# Continuous attributes (2)

| Temp. | Play |
|---|---|
| 80 | No |
| 85 | No |
| 83 | Yes |
| 75 | Yes |
| 68 | Yes |
| 65 | No |
| 64 | Yes |
| 72 | No |
| 75 | Yes |
| 70 | Yes |
| 69 | Yes |
| 72 | Yes |
| 81 | Yes |
| 71 | No |

Sort →

| Temp. | Play |
|---|---|
| 64 | Yes |
| 65 | No |
| 68 | Yes |
| 69 | Yes |
| 70 | Yes |
| 71 | No |
| 72 | No |
| 72 | Yes |
| 75 | Yes |
| 75 | Yes |
| 80 | No |
| 81 | Yes |
| 83 | Yes |
| 85 | No |

Temp.< 64.5 ΔI=0.048

Temp.< 66.5 ΔI=0.010

Temp.< 68.5 ΔI=0.000

Temp.< 69.5 ΔI=0.015

Temp.< 70.5 ΔI=0.045

Temp.< 71.5 ΔI=0.001

Temp.< 73.5 ΔI=0.001

Temp.< 77.5 ΔI=0.025

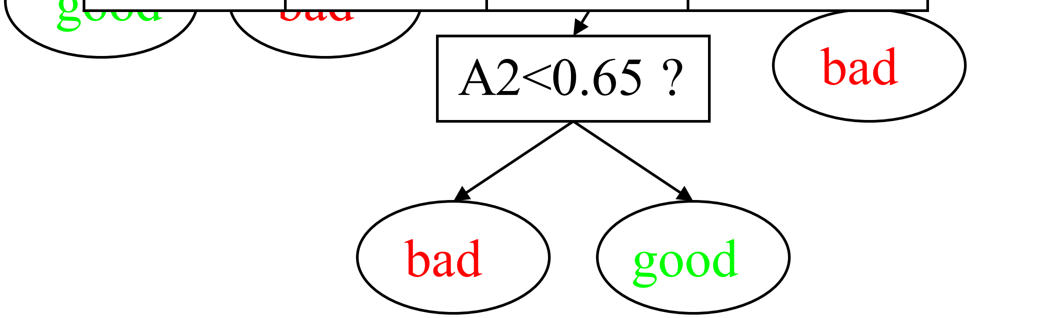
Temp.< 80.5 ΔI=0.000

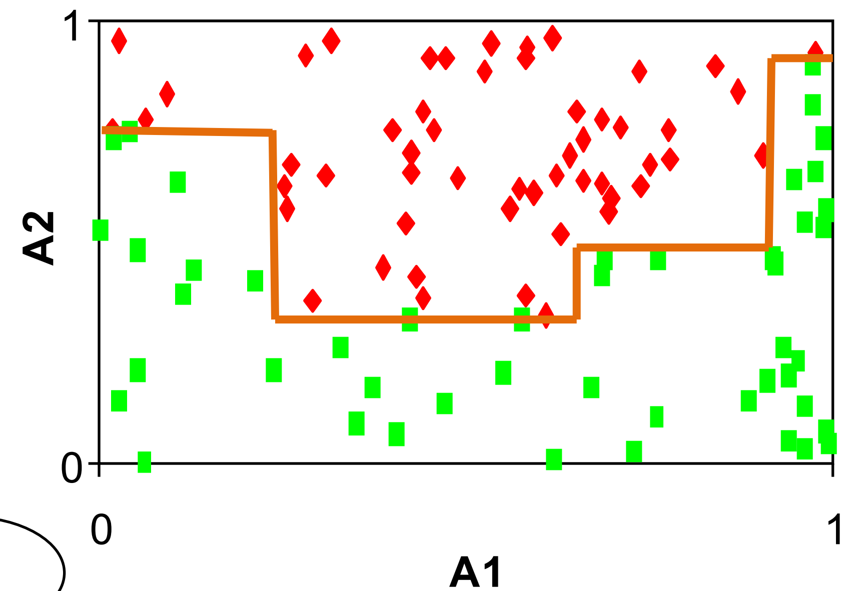
Temp.< 82 ΔI=0.010

Temp.< 84 ΔI=0.113

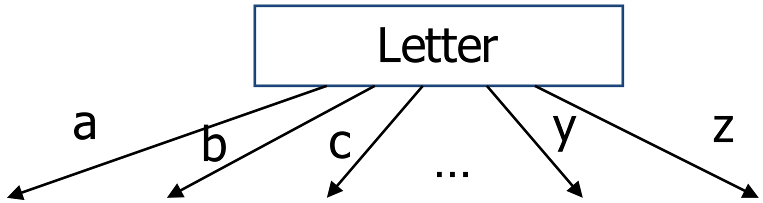
# Continuous attribute (3)

| Number | A1 | A2 | Colour |
|---|---|---|---|
| 1 | 0.58 | 0.75 | Red |
| 2 | 0.78 | 0.65 | Red |
| 3 | 0.89 | 0.23 | Green |
| 4 | 0.12 | 0.98 | Red |
| 5 | 0.17 | 0.26 | Green |
| 6 | 0.50 | 0.48 | Red |
| 7 | 0.45 | 0.16 | Green |
| 8 | 0.80 | 0.75 | Green |
| … | … | … | … |
| 100 | 0.75 | 0.13 | Green |



# Attributes with many values (1)




- Problem:
  - Not good splits: they fragment the data too quickly, leaving insufficient data at the next level
  - The reduction of impurity of such test is often high (example: split on the object id).
- Two solutions:
  - Change the splitting criterion to penalize attributes with many values
  - Consider only binary splits (preferable)

# Attributes with many values (2)

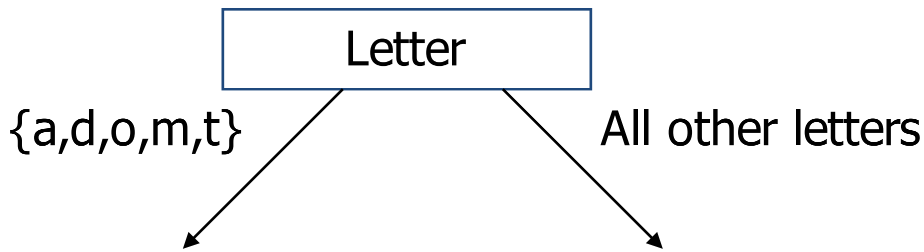
- Modified splitting criterion:
  - Gainratio($LS$,$A$)= $\Delta H(LS,A)$/Splitinformation($LS$,$A$)
  - Splitinformation($LS$,$A$)=$-\sum_a |LS_a|/|LS| \log(|LS_a|/|LS|)$
  - The split information is high when there are many values
- Example: outlook in the playtennis
  - $\Delta H(LS,\text{outlook}) = 0.246$
  - Splitinformation($LS$,outlook)= 1.577
  - Gainratio($LS$,outlook) = $0.246/1.577=0.156 < 0.246$
- Problem: the gain ratio favours unbalanced tests

# A simple database: playtennis

| Day | Outlook | Temperature | Humidity | Wind | Play Tennis |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | Normal | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | High | Strong | Yes |
| D8 | Sunny | Mild | Normal | Weak | No |
| D9 | Sunny | Hot | Normal | Weak | Yes |
| D10 | Rain | Mild | Normal | Strong | Yes |
| D11 | Sunny | Cool | Normal | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |
| D14 | Rain | Mild | High | Strong | No |

# Attributes with many values (3)

- Allow binary tests only:




- There are $2^N$-1 possible subsets for *N* values
- If *N* is small, determination of the best subsets by enumeration
- If *N* is large, heuristics exist (e.g. greedy approach)

# Missing attribute values

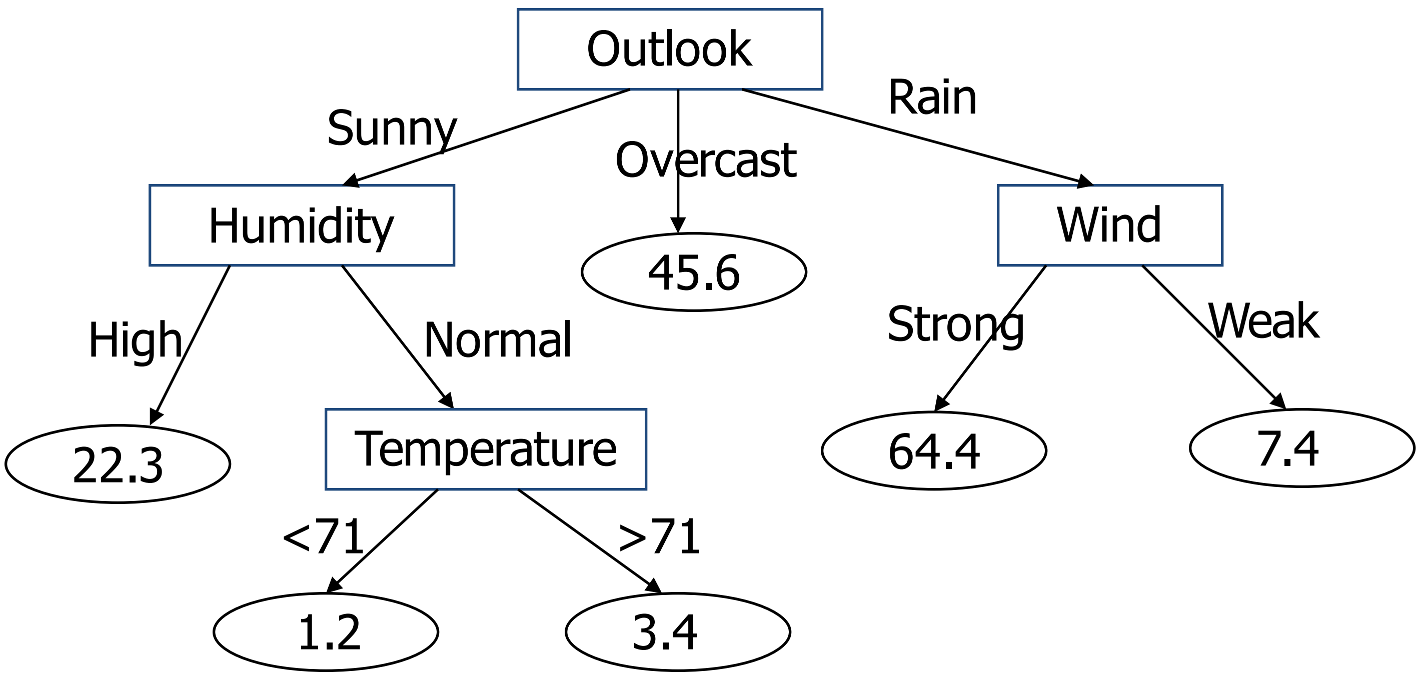
- Not all attribute values known for every objects when learning or when testing

| Day | Outlook | Temperature | Humidity | Wind | Play Tennis |
| --- | --- | --- | --- | --- | --- |
| D15 | Sunny | Hot | ? | Strong | No |

- Three strategies:
  - Assign most common value in the learning sample
  - Assign most common value in tree
  - Assign probability to each possible value

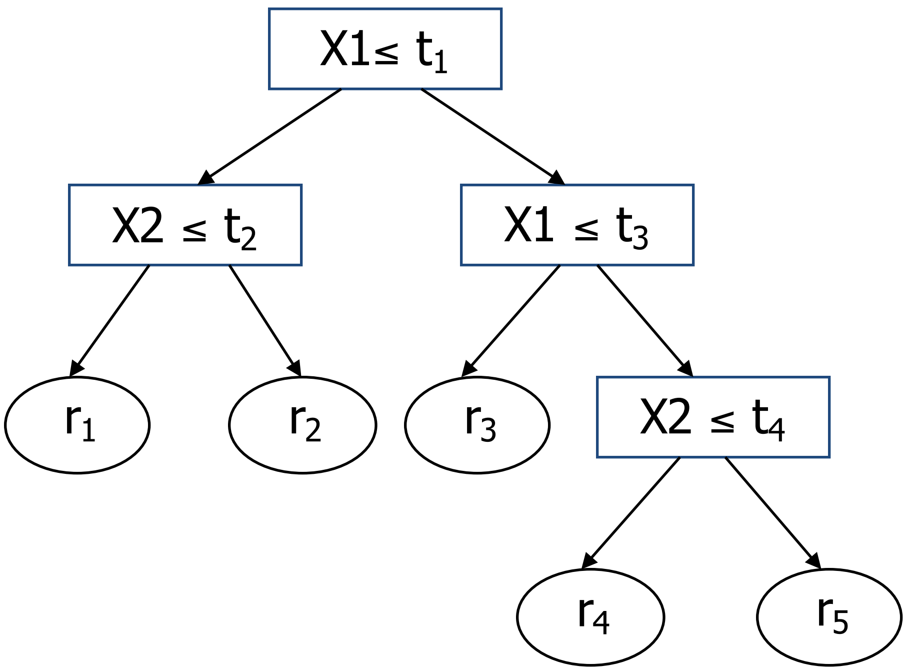
# Regression trees (1)

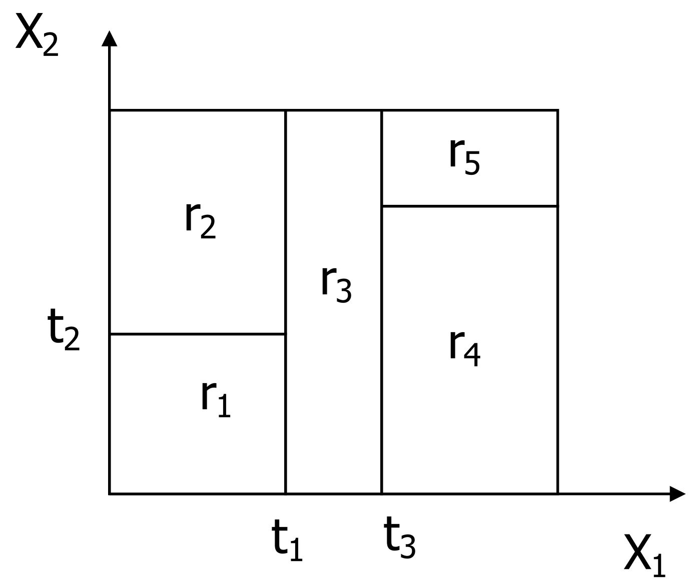
- Tree for regression: exactly the same model but with a number in each leaf instead of a class



# Regression trees (2)

- A regression tree is a piecewise constant function of the input attributes



# Regression tree growing

- To minimize the square error on the learning sample, the prediction at a leaf is the average output of the learning cases reaching that leaf
- Impurity of a sample is defined by the variance of the output in that sample:

$$I(LS)=\mathrm{var}_{y|LS}\{y\}=\mathrm{E}_{y|LS}\{(y\text{-}\mathrm{E}_{y|LS}\{y\})^2\}$$

- The best split is the one that reduces the most variance:

$$\Delta I(LS, A) = \mathrm{var}_{y|LS}\{y\} - \sum_a \frac{|LS_a|}{|LS|} \mathrm{var}_{y|LS_a}\{y\}$$

# Regression tree pruning

- Exactly the same algorithms apply: pre-pruning and post-pruning.
- In post-pruning, the tree that minimizes the squared error on $VS$ is selected.

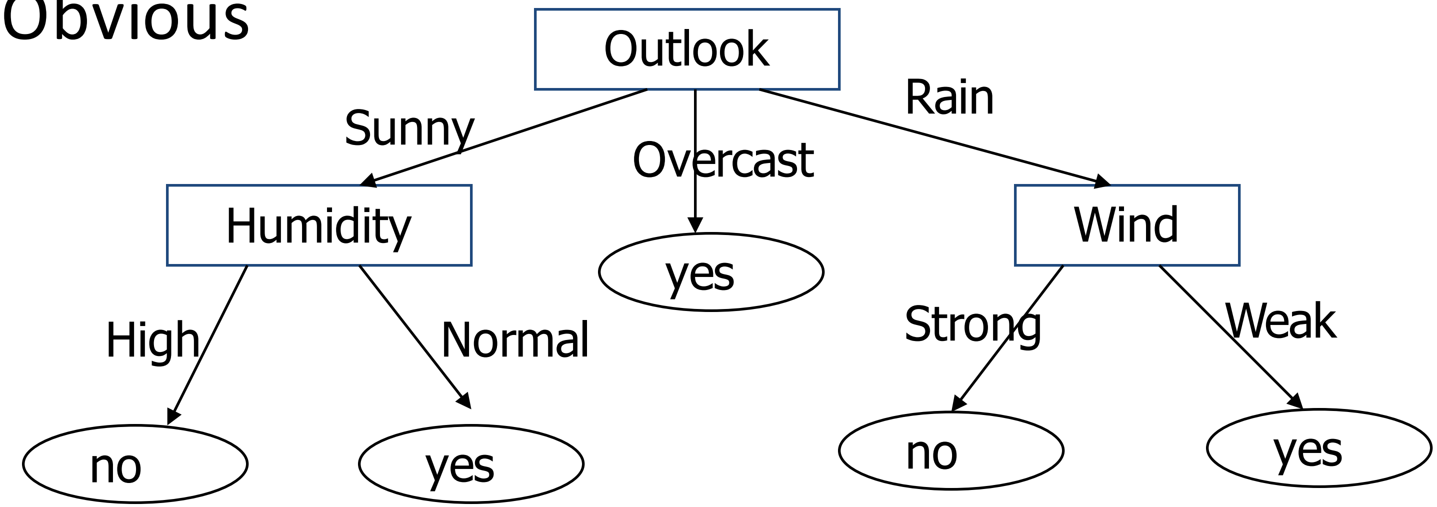
- In practice, pruning is more important in regression because full trees are much more complex (often all objects have a different output values and hence the full tree has as many leaves as there are objects in the learning sample)

# Outline

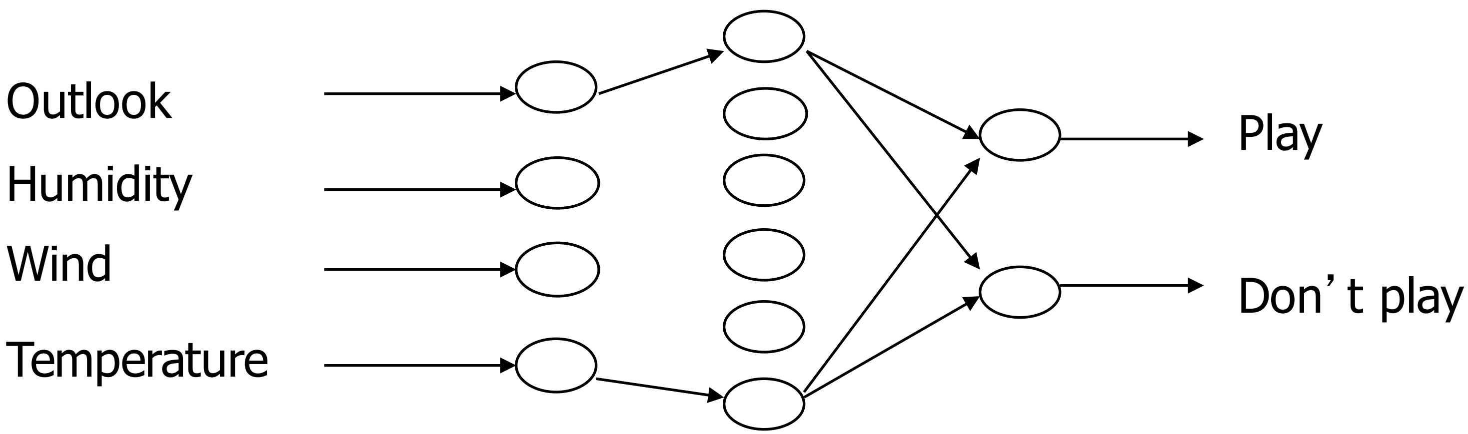
- Supervised learning
- Tree representation
- Tree learning
- Extensions
- Regression trees
- By-products
  - Interpretability
  - Variable selection
  - Variable importance

# Interpretability (1)

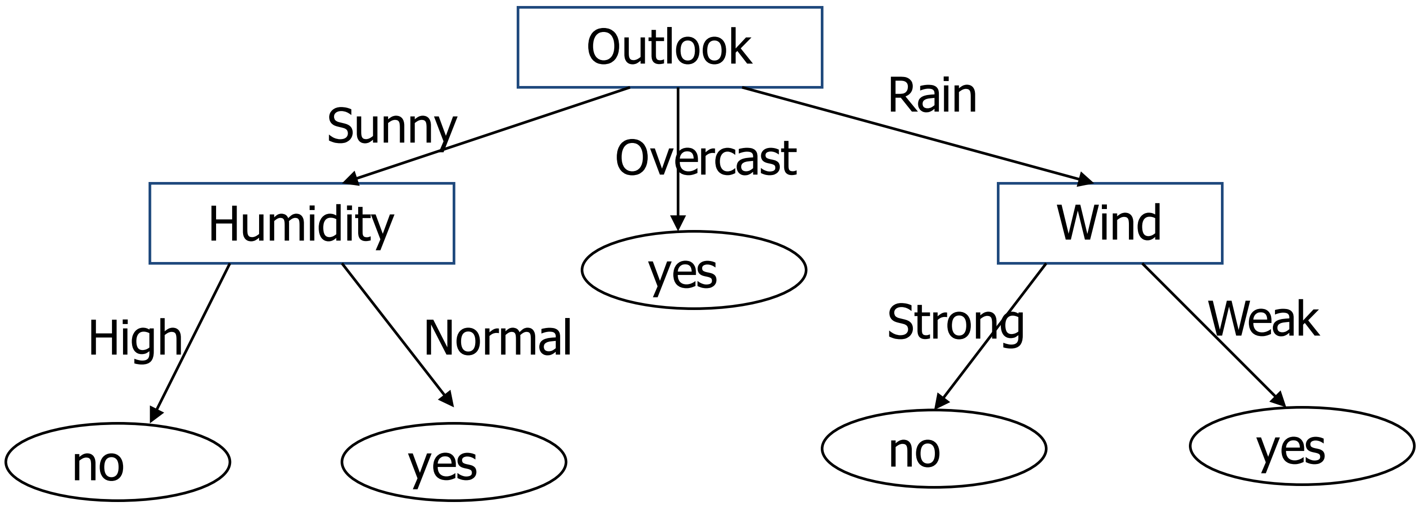
- Obvious




- Compare with a neural networks:



# Interpretability (2)



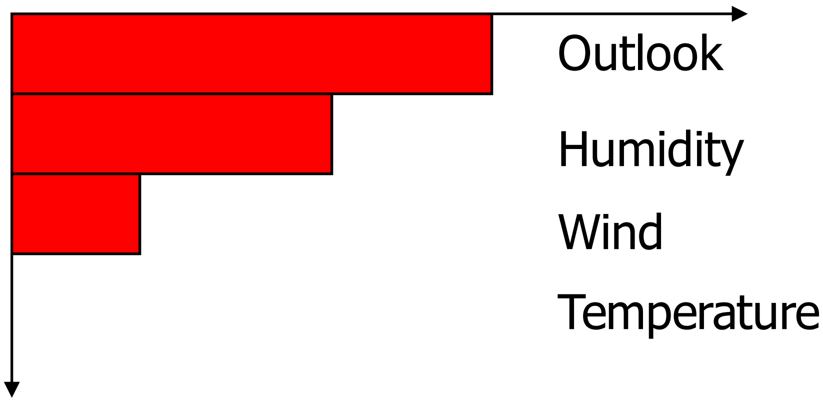

- A tree may be converted into a set of rules
  - If (outlook=sunny) and (humidity=high) then PlayTennis=No
  - If (outlook=sunny) and (humidity=normal) then PlayTennis=Yes
  - If (outlook=overcast) then PlayTennis=Yes
  - If (outlook=rain) and (wind=strong) then PlayTennis=No
  - If (outlook=rain) and (wind=weak) then PlayTennis=Yes

# Attribute selection

- If some attributes are not useful for classification, they will not be selected in the (pruned) tree
- Of practical importance, if measuring the value of an attribute is costly (e.g. medical diagnosis)
- Decision trees are often used as a pre-processing for other learning algorithms that suffer more when there are irrelevant variables

# Variable importance

- In many applications, all variables do not contribute equally in predicting the output.
- We can evaluate variable importance by computing the total reduction of impurity brought by each variable:
  - $\text{Imp}(A)=\sum_{\text{nodes where } A \text{ is tested}} |LS_{\text{node}}|\ \Delta I(LS_{\text{node}},A)$



# When are decision trees useful ?

- Advantages
  - Very fast: can handle very large datasets with many attributes
    - Complexity $O(n.N \log N)$, with n the number of attributes and N the number of samples.
  - Flexible: several attribute types, classification and regression problems, missing values…
  - Interpretability: provide rules and attribute importance
- Disadvantages
  - Instability of the trees (high variance)
  - Not always competitive with other algorithms in terms of accuracy

# Further extensions and research

- Cost and un-balanced learning sample
- Oblique trees (test like $\sum \alpha_i A_i < a_{th}$)
- Using predictive models in leaves (e.g. linear regression)
- Induction graphs
- Fuzzy decision trees (from a crisp partition to a fuzzy partition of the learning sample)

# Demo

- Illustration on two datasets:
  - titanic
    - http://www.cs.toronto.edu/~delve/data/titanic/desc.html
  - splice junction
    - http://www.cs.toronto.edu/~delve/data/splice/desc.html

# References

- Chapter 9, Section 9.2 of the reference book (Hastie et al., 2009).
- Understanding random forests, Gilles Louppe, PhD thesis, Ulg, 2014 (http://hdl.handle.net/2268/170309)
- Supplementary slides are also available on the course website

- *Classification and regression trees*, L.Breiman et al., Wadsworth, 1984
- *C4.5: programs for machine learning*, J.R.Quinlan, Morgan Kaufmann, 1993
- *Graphes d'induction*, D.Zighed and R.Rakotomalala, Hermes, 2000

# Softwares

- scikit-learn:
  - http://scikit-learn.org
- Weka
  - J48
  - http://www.cs.waikato.ac.nz/ml/weka
- In R:
  - Packages tree and rpart