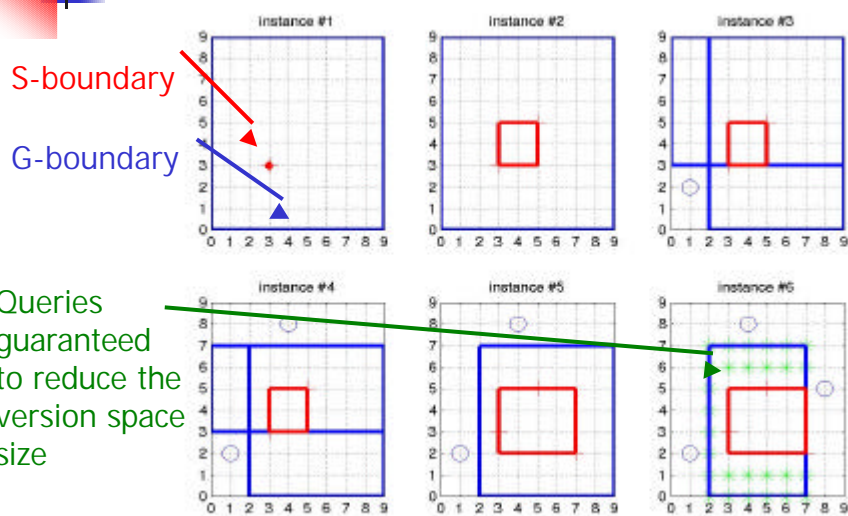


Machine Learning 2D5362

Lecture 3: Decision Tree Learning

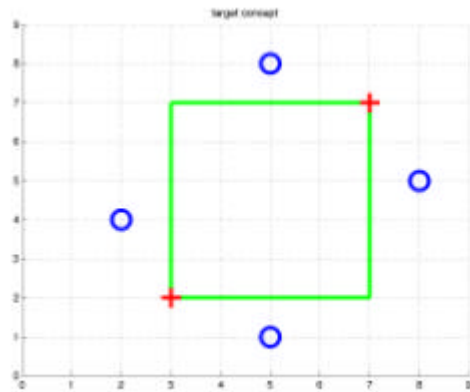
Homework: Concept Learner





Homework: Concept Learner

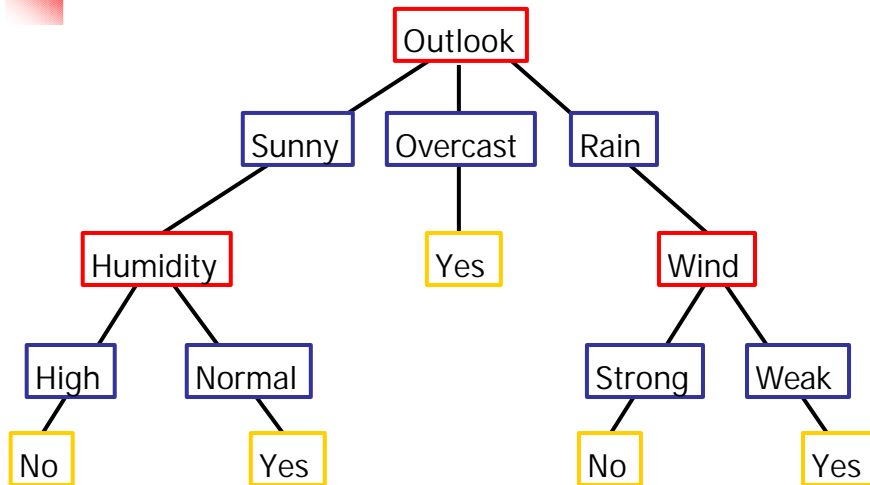
Ideal training examples to teach the target concept



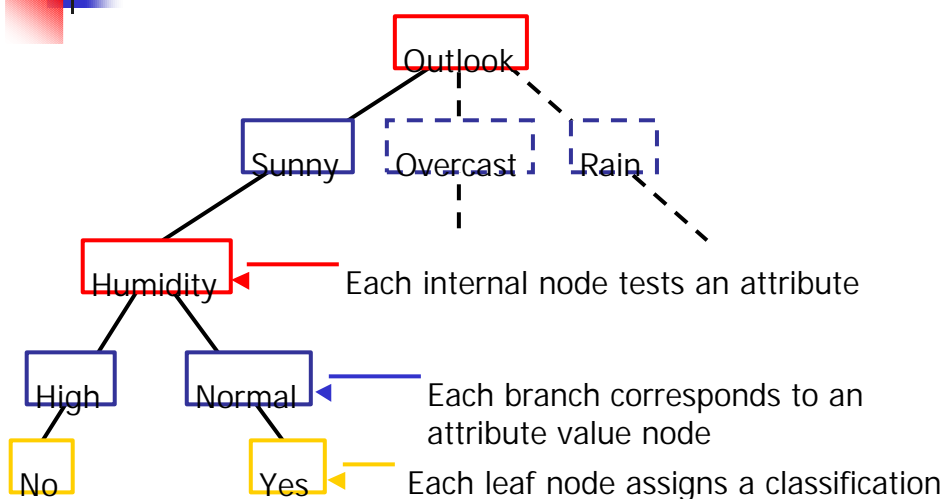
Outline

- ✍ Decision tree representation
- ✍ ID3 learning algorithm
- ✍ Entropy, information gain
- ✍ Overfitting

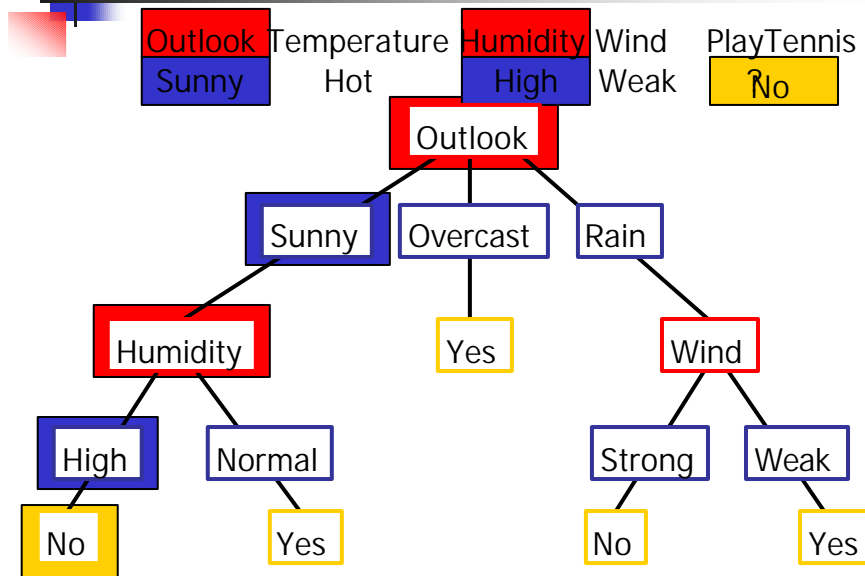
Decision Tree for PlayTennis



Decision Tree for PlayTennis

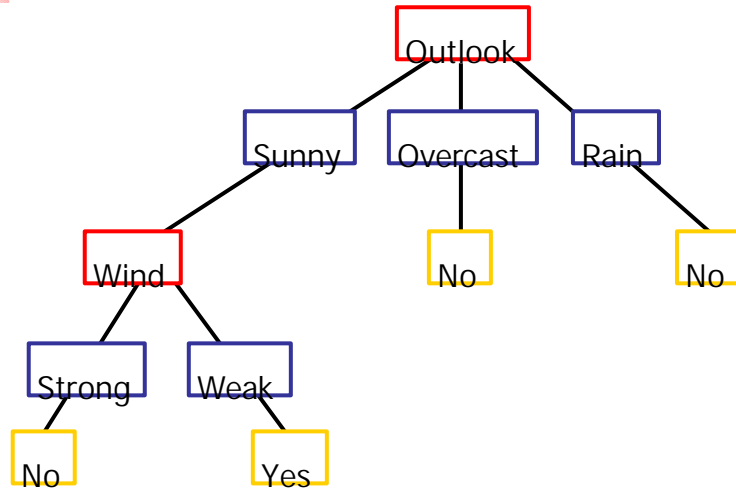


Decision Tree for PlayTennis



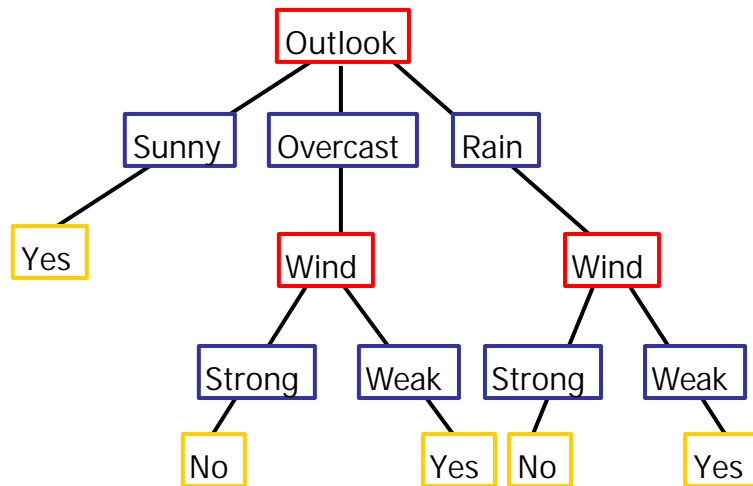
Decision Tree for Conjunction

Outlook=Sunny ? Wind=Weak



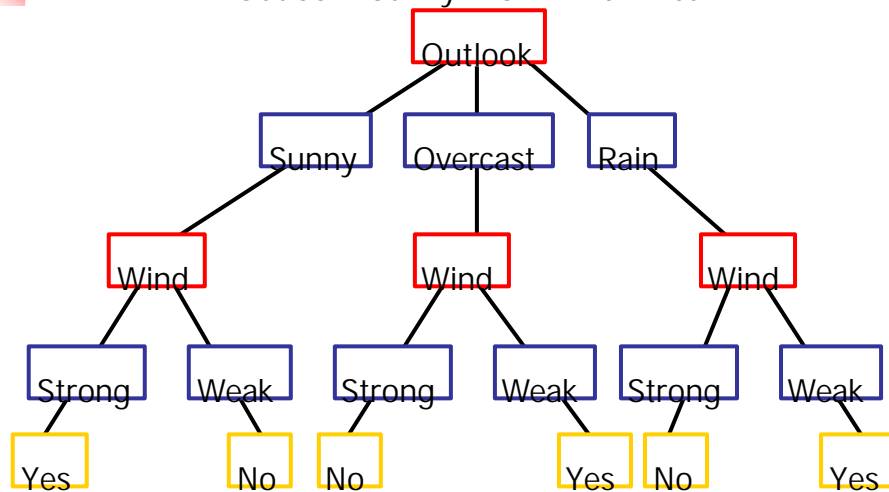
Decision Tree for Disjunction

Outlook=Sunny ? Wind=Weak



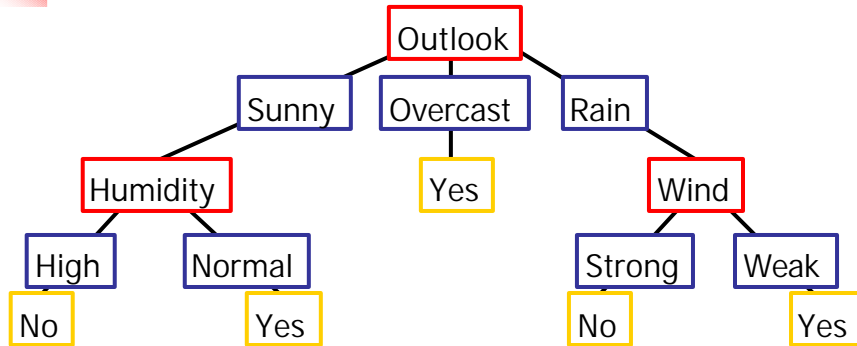
Decision Tree for XOR

Outlook=Sunny XOR Wind=Weak



Decision Tree

- decision trees represent disjunctions of conjunctions



(Outlook=Sunny ? Humidity=Normal)
? (Outlook=Overcast)
? (Outlook=Rain ? Wind=Weak)

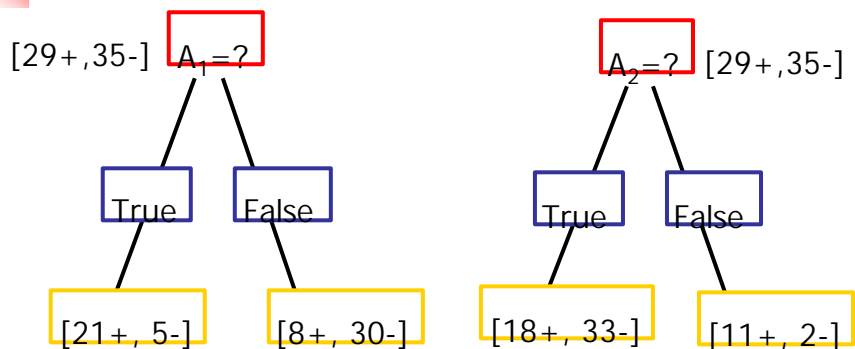
When to consider Decision Trees

- ✗ Instances describable by attribute-value pairs
- ✗ Target function is discrete valued
- ✗ Disjunctive hypothesis may be required
- ✗ Possibly noisy training data
- ✗ Missing attribute values
- ✗ Examples:
 - ✗ Medical diagnosis
 - ✗ Credit risk analysis
 - ✗ Object classification for robot manipulator (Tan 1993)

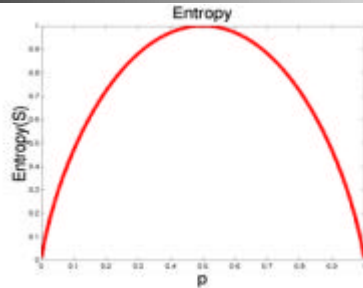
Top-Down Induction of Decision Trees ID3

1. A ? the "best" decision attribute for next *node*
2. Assign A as decision attribute for *node*
3. For each value of A create new descendant
4. Sort training examples to leaf node according to the attribute value of the branch
5. If all training examples are perfectly classified (same value of target attribute) stop, else iterate over new leaf nodes.

Which Attribute is "best"?



Entropy



- ⌘ S is a sample of training examples
 - ⌘ p_+ is the proportion of positive examples
 - ⌘ p_- is the proportion of negative examples
 - ⌘ Entropy measures the impurity of S
- $$\text{Entropy}(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

Entropy

- ⌘ Entropy(S) = expected number of bits needed to encode class (+ or -) of randomly drawn members of S (under the optimal, shortest length-code)

Why?

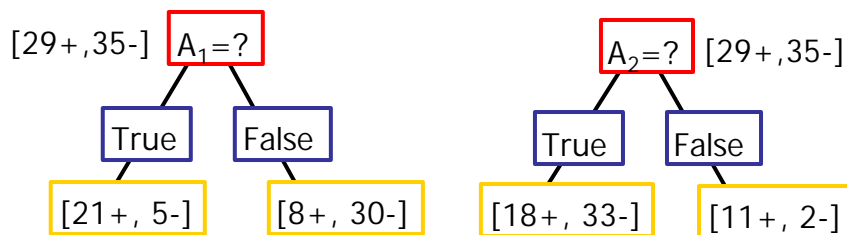
- ⌘ Information theory optimal length code assign $-\log_2 p$ bits to messages having probability p.
- ⌘ So the expected number of bits to encode (+ or -) of random member of S:
$$-p_+ \log_2 p_+ - p_- \log_2 p_-$$

Information Gain

Gain(S,A): expected reduction in entropy due to sorting S on attribute A

$$\text{Gain}(S,A) = \text{Entropy}(S) - \sum_{v \in \text{values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

$$\text{Entropy}([29+, 35-]) = -29/64 \log_2 29/64 - 35/64 \log_2 35/64 = 0.99$$



Information Gain

$$\text{Entropy}([21+, 5-]) = 0.71$$

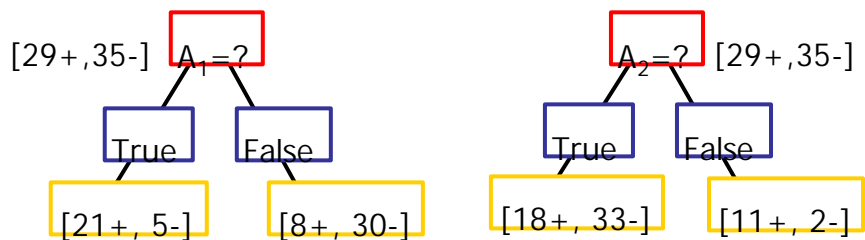
$$\text{Entropy}([8+, 30-]) = 0.74$$

$$\begin{aligned} \text{Gain}(S,A_1) &= \text{Entropy}(S) \\ &\quad - 26/64 * \text{Entropy}([21+, 5-]) \\ &\quad - 38/64 * \text{Entropy}([8+, 30-]) \\ &= 0.27 \end{aligned}$$

$$\text{Entropy}([18+, 33-]) = 0.94$$

$$\text{Entropy}([11+, 2-]) = 0.62$$

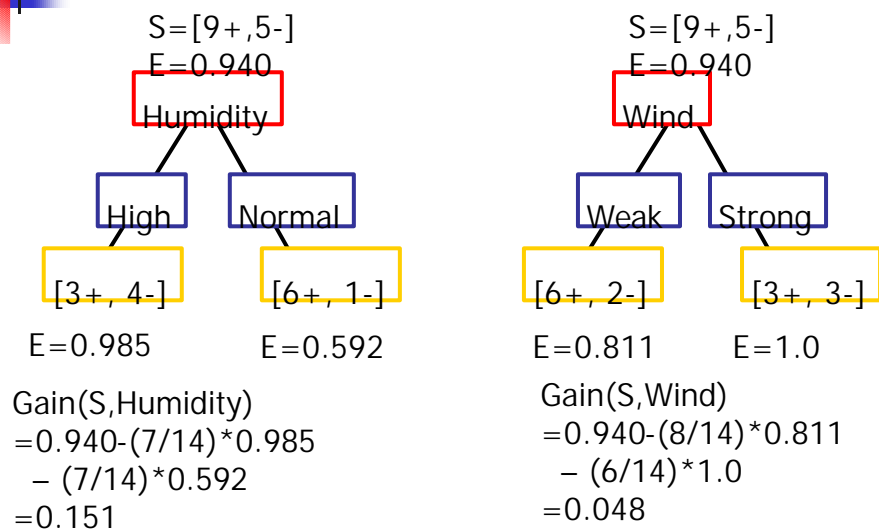
$$\begin{aligned} \text{Gain}(S,A_2) &= \text{Entropy}(S) \\ &\quad - 51/64 * \text{Entropy}([18+, 33-]) \\ &\quad - 13/64 * \text{Entropy}([11+, 2-]) \\ &= 0.12 \end{aligned}$$



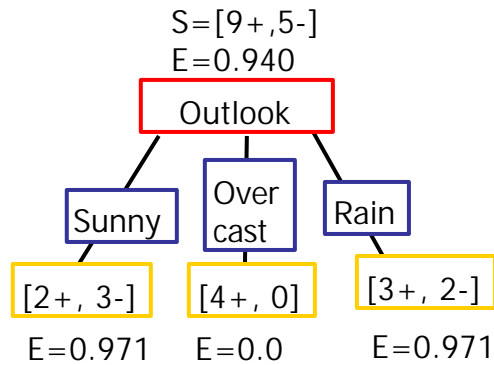
Training Examples

Day	Outlook	Temp.	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	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cold	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Selecting the Next Attribute

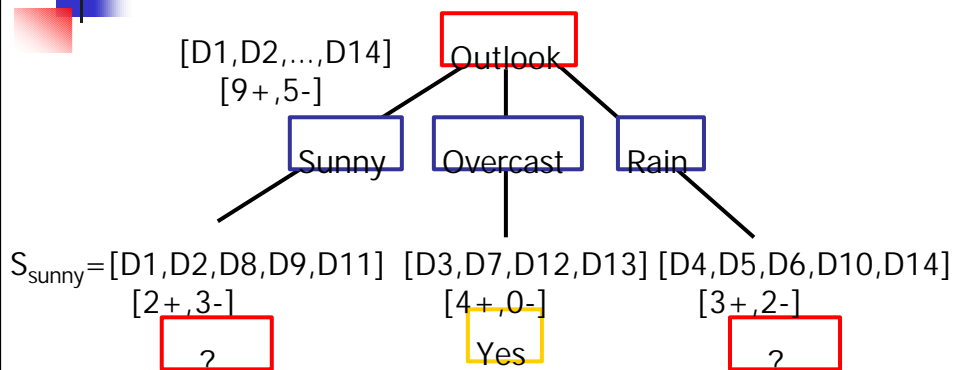


Selecting the Next Attribute



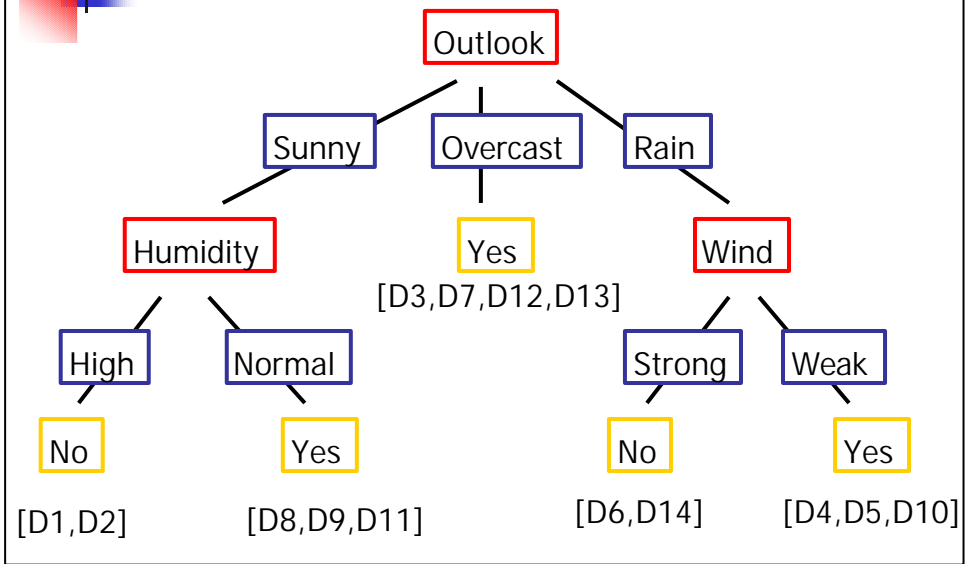
$$\begin{aligned}
 \text{Gain}(S, \text{Outlook}) &= 0.940 - (5/14) * 0.971 \\
 &\quad - (4/14) * 0.0 - (5/14) * 0.971 \\
 &= 0.247
 \end{aligned}$$

ID3 Algorithm

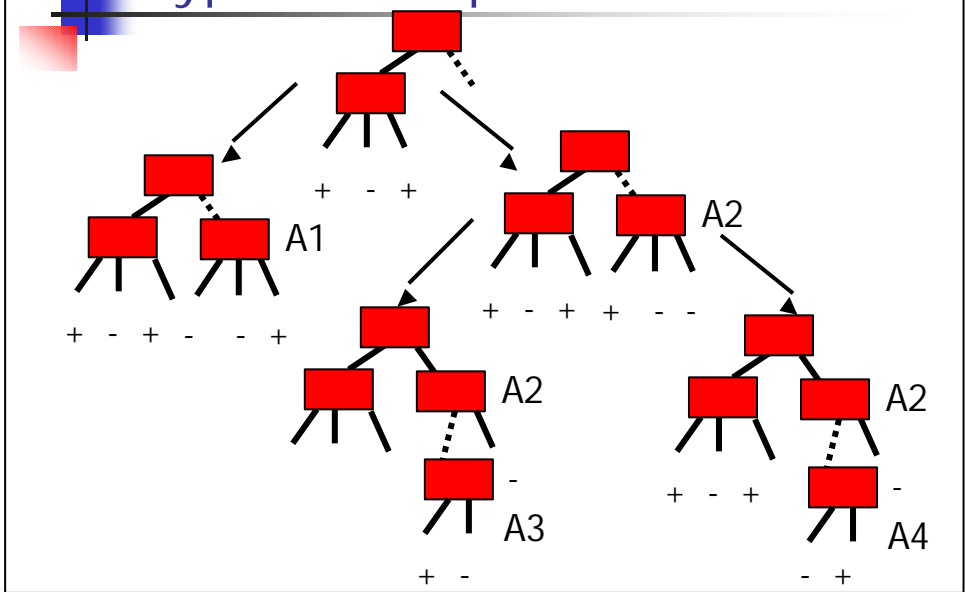


$$\begin{aligned}
 \text{Gain}(S_{\text{sunny}}, \text{Humidity}) &= 0.970 - (3/5)0.0 - 2/5(0.0) = 0.970 \\
 \text{Gain}(S_{\text{sunny}}, \text{Temp.}) &= 0.970 - (2/5)0.0 - 2/5(1.0) - (1/5)0.0 = 0.570 \\
 \text{Gain}(S_{\text{sunny}}, \text{Wind}) &= 0.970 - (2/5)1.0 - 3/5(0.918) = 0.019
 \end{aligned}$$

ID3 Algorithm



Hypothesis Space Search ID3






Hypothesis Space Search ID3

- ⌘ Hypothesis space is complete!
 - ⌘ Target function surely in there...
- ⌘ Outputs a single hypothesis
- ⌘ No backtracking on selected attributes (greedy search)
 - ⌘ Local minimal (suboptimal splits)
- ⌘ Statistically-based search choices
 - ⌘ Robust to noisy data
- ⌘ Inductive bias (search bias)
 - ⌘ Prefer shorter trees over longer ones
 - ⌘ Place high information gain attributes close to the root



Inductive Bias in ID3

- ⌘ H is the power set of instances X
 - ⌘ Unbiased ?
- ⌘ Preference for short trees, and for those with high information gain attributes near the root
- ⌘ Bias is a *preference* for some hypotheses, rather than a *restriction* of the hypothesis space H
- ⌘ Occam's razor: prefer the shortest (simplest) hypothesis that fits the data



Occam's Razor


Why prefer short hypotheses?

Argument in favor:

- ⚡ Fewer short hypotheses than long hypotheses
- ⚡ A short hypothesis that fits the data is unlikely to be a coincidence
- ⚡ A long hypothesis that fits the data might be a coincidence

Argument opposed:

- ⚡ There are many ways to define small sets of hypotheses
- ⚡ E.g. All trees with a prime number of nodes that use attributes beginning with "Z"
- ⚡ What is so special about small sets based on size of hypothesis



Occam's Razor

Definition A: 

green 

blue 

Hypothesis A:

Objects do not instantaneously change their color.

Definition B:

grue 

bleen 

Hypothesis B:

1.1.2000 0:00

On 1.1.2000 objects that were grue turned instantaneously bleen and objects that were bleen turned instantaneously grue.



Overfitting

Consider error of hypothesis h over

- ✗ Training data: $\text{error}_{\text{train}}(h)$
- ✗ Entire distribution D of data: $\text{error}_D(h)$

Hypothesis h ? H *overfits* training data if there is an alternative hypothesis h' such that

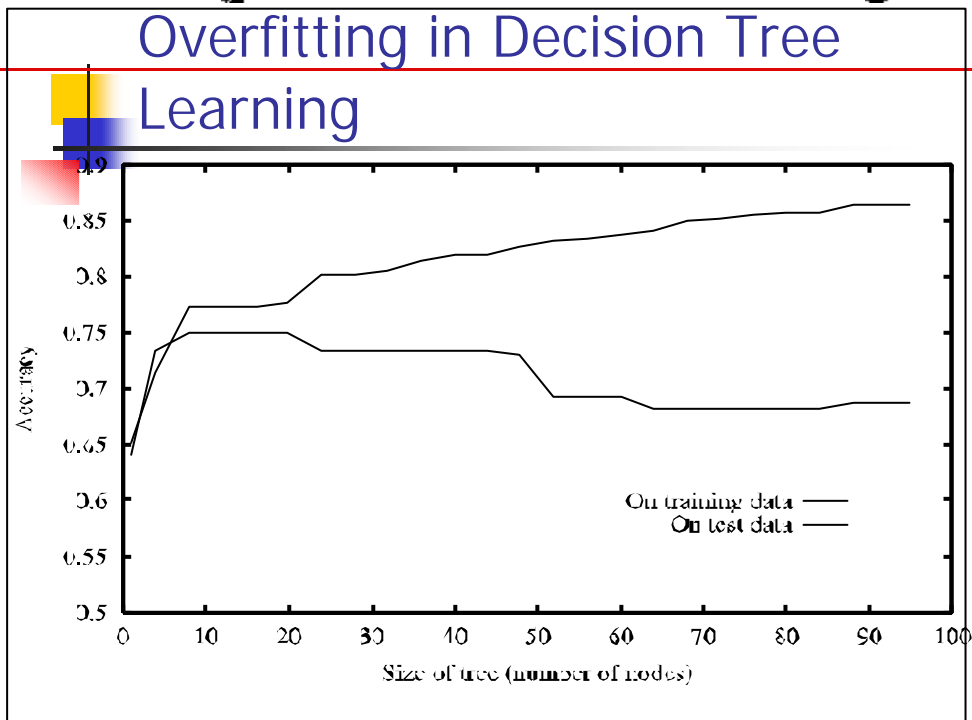
$$\text{error}_{\text{train}}(h) < \text{error}_{\text{train}}(h')$$

and

$$\text{error}_D(h) > \text{error}_D(h')$$

Overfitting in Decision Tree Learning

Overfitting in Decision Tree Learning





Avoid Overfitting

How can we avoid overfitting?

- ✗ Stop growing when data split not statistically significant
- ✗ Grow full tree then post-prune
- ✗ Minimum description length (MDL):
Minimize:
 $\text{size}(\text{tree}) + \text{size}(\text{misclassifications}(\text{tree}))$



Reduced-Error Pruning

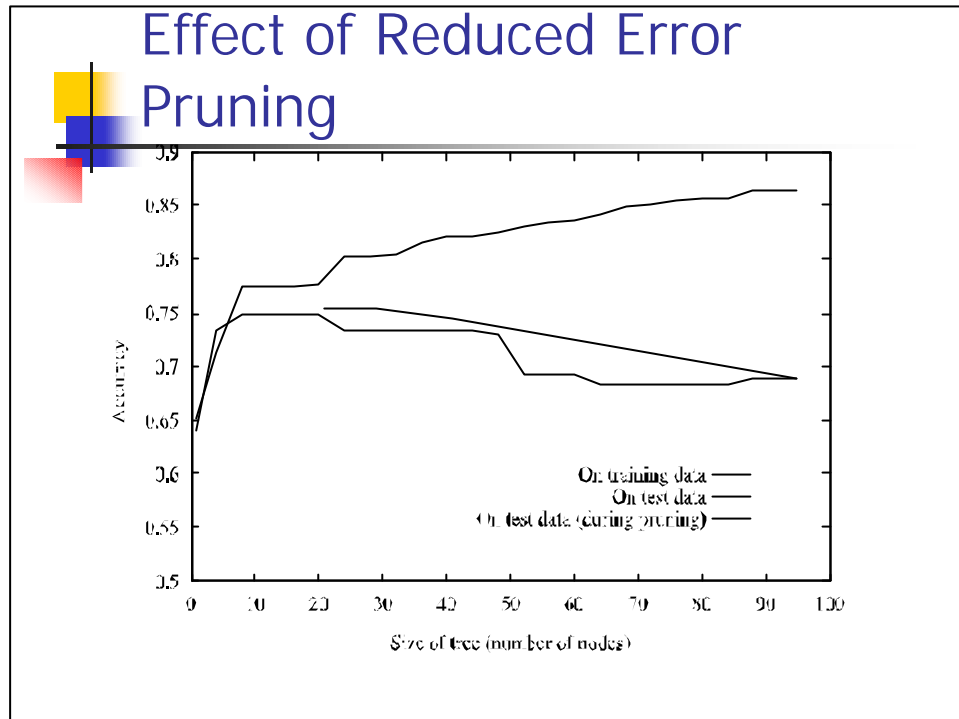
Split data into *training* and validation set

Do until further pruning is harmful:

1. Evaluate impact on *validation* set of pruning each possible node (plus those below it)
2. Greedily remove the one that most improves the *validation* set accuracy

Produces smallest version of most accurate subtree

Effect of Reduced-Error Pruning

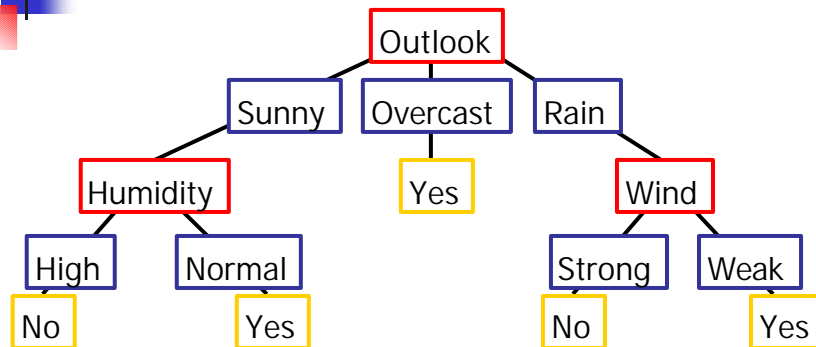


Rule-Post Pruning

1. Convert tree to equivalent set of rules
2. Prune each rule independently of each other
3. Sort final rules into a desired sequence to use

Method used in C4.5

Converting a Tree to Rules



- R_1 : If (Outlook=Sunny) ? (Humidity=High) Then PlayTennis=No
 R_2 : If (Outlook=Sunny) ? (Humidity=Normal) Then PlayTennis=Yes
 R_3 : If (Outlook=Overcast) Then PlayTennis=Yes
 R_4 : If (Outlook=Rain) ? (Wind=Strong) Then PlayTennis=No
 R_5 : If (Outlook=Rain) ? (Wind=Weak) Then PlayTennis=Yes

Continuous Valued Attributes

Create a discrete attribute to test continuous

- ⌘ Temperature = 24.5°C
- ⌘ (Temperature > 20.0°C) = {true, false}

Where to set the threshold?

Temperatur	15°C	18°C	19°C	22°C	24°C	27°C
PlayTennis	No	No	Yes	Yes	Yes	No

(see paper by [Fayyad, Irani 1993])



Attributes with many Values

⚡ Problem: if an attribute has many values, maximizing *InformationGain* will select it.

⚡ E.g.: Imagine using Date=12.7.1996 as attribute perfectly splits the data into subsets of size 1

Use *GainRatio* instead of information gain as criteria:

$$\text{GainRatio}(S,A) = \text{Gain}(S,A) / \text{SplitInformation}(S,A)$$

$$\text{SplitInformation}(S,A) = -\sum_{i=1..c} |S_i|/|S| \log_2 |S_i|/|S|$$

Where S_i is the subset for which attribute A has the value v_i



Attributes with Cost

Consider:

⚡ Medical diagnosis : blood test costs 1000 SEK

⚡ Robotics: width_from_one_foot has cost 23 secs.

How to learn a consistent tree with low expected cost?

Replace *Gain* by :

$$\text{Gain}^2(S,A)/\text{Cost}(A) \quad [\text{Tan, Schimmer 1990}]$$

$$2^{\text{Gain}(S,A)} - 1 / (\text{Cost}(A) + 1)^w \quad w ? [0,1] \quad [\text{Nunez 1988}]$$



Unknown Attribute Values

What is some examples missing values of A?

Use training example anyway sort through tree

- ⌘ If node n tests A , assign most common value of A among other examples sorted to node n .
- ⌘ Assign most common value of A among other examples with same target value
- ⌘ Assign probability p_i to each possible value v_i of A
 - ⌘ Assign fraction p_i of example to each descendant in tree

Classify new examples in the same fashion



Cross-Validation

- ⌘ Estimate the accuracy of a hypothesis induced by a supervised learning algorithm
- ⌘ Predict the accuracy of a hypothesis over future unseen instances
- ⌘ Select the optimal hypothesis from a given set of alternative hypotheses
 - ⌘ Pruning decision trees
 - ⌘ Model selection
 - ⌘ Feature selection
- ⌘ Combining multiple classifiers (boosting)



Holdout Method

Partition data set $D = \{(v_1, y_1), \dots, (v_n, y_n)\}$ into *training* D_t and *validation* set $D_h = D \setminus D_t$



$$\text{acc}_h = \frac{1}{n} \sum_{(v_i, y_i) \in D_h} \mathbb{1}(I(D_t, v_i) = y_i)$$

$I(D_t, v_i)$: output of hypothesis induced by learner I trained on data D_t for instance v_i

$\mathbb{1}(i, j) = 1$ if $i=j$ and 0 otherwise

Problems:

- makes insufficient use of data
- training and validation set are correlated

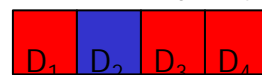
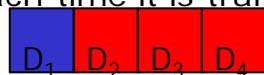


Cross-Validation

k-fold cross-validation splits the data set D into k mutually exclusive subsets D_1, D_2, \dots, D_k



Train and test the learning algorithm k times, each time it is trained on $D \setminus D_i$ and tested on D_i



$$\text{acc}_{cv} = \frac{1}{n} \sum_{(v_i, y_i) \in D} \mathbb{1}(I(D \setminus D_i, v_i) = y_i)$$



Cross-Validation

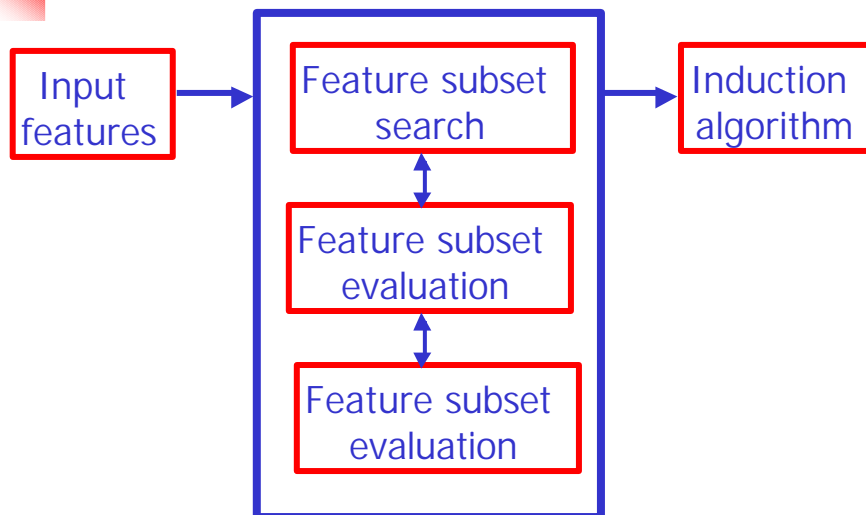
- ✎ Uses all the data for training and testing
- ✎ Complete k-fold cross-validation splits the dataset of size m in all $\binom{m}{m/k}$ possible ways (choosing m/k instances out of m)
- ✎ Leave n -out cross-validation sets n instances aside for testing and uses the remaining ones for training (leave one-out is equivalent to n -fold cross-validation)
- ✎ In stratified cross-validation, the folds are stratified so that they contain approximately the same proportion of labels as the original data set



Bootstrap

- ✎ Samples n instances uniformly from the data set with replacement
- ✎ Probability that any given instance is not chosen after n samples is $(1-1/n)^n \approx e^{-1} \approx 0.632$
- ✎ The bootstrap sample is used for training the remaining instances are used for testing
- ✎ $acc_{boot} = 1/b \sum_{i=1}^b (0.632 \theta_i + 0.368 acc_s)$
where θ_i is the accuracy on the test data of the i -th bootstrap sample, acc_s is the accuracy estimate on the training set and b the number of bootstrap samples

Wrapper Model

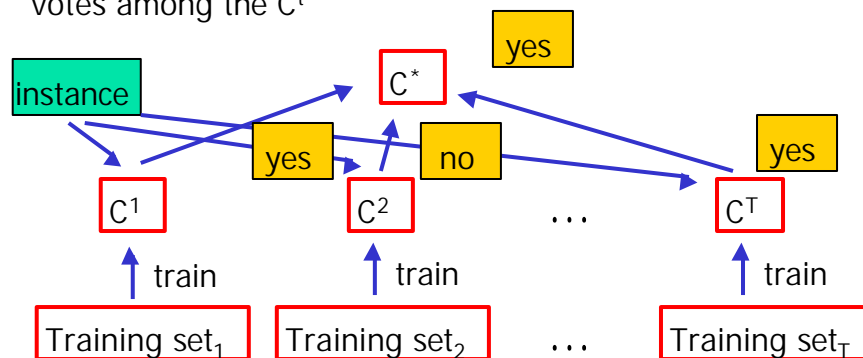


Wrapper Model

- ✧ Evaluate the accuracy of the inducer for a given subset of features by means of n-fold cross-validation
- ✧ The training data is split into n folds, and the induction algorithm is run n times. The accuracy results are averaged to produce the estimated accuracy.
- ✧ Forward elimination:
 - Starts with the empty set of features and greedily adds the feature that improves the estimated accuracy at most
- ✧ Backward elimination:
 - Starts with the set of all features and greedily removes features and greedily removes the worst feature

Bagging

- For each trial $t=1,2,\dots,T$ create a bootstrap sample of size N .
- Generate a classifier C^t from the bootstrap sample
- The final classifier C^* takes class that receives the majority votes among the C^t



Bagging

- Bagging requires "instable" classifiers like for example decision trees or neural networks

"The vital element is the instability of the prediction method. If perturbing the learning set can cause significant changes in the predictor constructed, then bagging can improve accuracy." (Breiman 1996)



MLC++

- ⌘ Machine Learning library in C++ developed by Ron Kohavi et al
- ⌘ Includes decision trees (ID3), decision graphs, nearest-neighbors, probabilities (Naive-Bayes), perceptron
- ⌘ Provides interfaces for external (not part of MLC++) inducers such as C4.5, OC1, PEBLS, CN2
- ⌘ Includes a set of tools (wrappers) for producing learning curves, perform accuracy estimation, generate statistics, bagging, feature subset selection



MLC++

- ⌘ MLC++ comes in two versions
 - ⌘ MLC++ utilities 2.0: compiled code for SUN takes options from the command line or environment variables
 - ⌘ MLC++ source code: you build the libraries and use the classes and methods in your own program
- ⌘ Documentation:
 - ⌘ Manual for the MLC++ Utilities : decent
 - ⌘ Tutorial for MLC++ libraries: minimal, you find some documentation in the source code itself



MLC++

Preliminaries:

- ⌘ Use gcc 2.7.2.3 not gcc 2.95
 - ⌘ module add gcc/2.7.2.3
- ⌘ MLC++ install directories
 - ⌘ /afs/nada.kth.se/home/cvap/hoffmann/Teaching/ML/MLC
 - ⌘ setenv MLCDIR /afs/nada.kth.se/home/cvap/hoffmann/Teaching/ML/MLC
 - ⌘ source \$MLCDIR/setup.SUN.GNU
 - ⌘ setenv PATH \$PATH\:\$MLCDIR/external/gnu\:\$MLCDIR/bin
- ⌘ Test installation
 - ⌘ cp \$MLCDIR/tutorial/dt1.c \$MLCDIR/tutorial/Makefile .
 - ⌘ make dt1
 - ⌘ ./dt1



Data Files

- ⌘ One dataset has three associated files:
 - ⌘ <dataset>.names : describes how to parse the data
 - ⌘ <dataset>.data : contains the training instances
 - ⌘ <dataset>.test : contains additional instances for estimating the accuracy
- ⌘ Monk1.names (| starts a comment)
 - no, yes
 - Head shape : round, square, octagon
 - Body shape : round, square, octagon
 - Holding : sword, ballon, flag
 - Jacket color: red, yellow, green, blue
 - Has tie: yes, no
- ⌘ Monk1.data
 - round, round, yes, sword, green, yes, yes
 - round, round, yes, sword, green, no, yes



Example ID3

```
#include <basics.h> // Must be included first in any program using MLC++
#include <ID3Inducer.h> // Include ID3Inducer class declarations
main()
{
    ID3Inducer inducer("my_first_ID3"); // "inducer" is an object of ID3Inducer
        // The argument is the "description."
    inducer.set_unknown_edges(FALSE); // Avoid edges for "unknown" attributes.
    inducer.read_data("monk"); // Read data files tutorial.names and
        // tutorial.data
    inducer.train(); // Produce categorizer
    inducer.display_struct(); // Display structure of categorizer

    return 0; // Return success to shell
}
```



Example Test Inducer

```
#include <ID3Inducer.h>
#include <PerfEstDispatch.h>
ID3Inducer inducer("my_first_ID3");
inducer.set_unknown_edges(FALSE);
inducer.read_data("monk");
Inducer.train();
InstanceList train("monk1");
InstanceList test(train.get_schema(),"monk1",".names",".test");
CatTestResult result(inducer.get_categorizer(), train, test);
result.display();

Classifying (% done): 10% 20% 30% 40% 50% 60% 70% 80%
Number of training instances: 124
Number of test instances: 432. Unseen: 308, seen 124.
Number correct: 350. Number incorrect: 82
Generalization error: 26.62%. Memorization error: 0.00%
Error: 18.98% +- 1.89% [15.56% - 22.95%]
Average Normalized Mean Squared Error: 18.98%
Average Normalized Mean Absolute Error: 18.98%
```



Example Bootstrap Estimation

```
#include <Bootstrap.h>
#include <ID3Inducer.h>
#include <PerfEstDispatch.h>

ID3Inducer inducer("my_first_ID3");
InstanceList train("monk1");
Bootstrap bootstrap(20);
bootstrap.init_rand_num_gen(0);
bootstrap.estimate_performance(inducer,train);
bootstrap.display();
```

Bootstrapping 20 times: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
13.30% +- 1.20% (2.81% - 22.05%)



Example Learning Curve

```
#include <LearnCurve.h>
#include <ID3Inducer.h>
ID3Inducer inducer("my_first_ID3");
LearnCurve learncurve;
learncurve.set_sizes_and_trials(7, 20, 1, 120, train);
learncurve.init_rand_num_gen(0);
learncurve.learn_curve(inducer,train);
```

Train Size: 1; training 20 times.....	Mean Error: 50.41% +- 0.00%
Train Size: 20; training 20 times.....	Mean Error: 35.82% +- 1.33%
Train Size: 40; training 20 times.....	Mean Error: 27.80% +- 1.68%
Train Size: 60; training 20 times.....	Mean Error: 26.25% +- 1.71%
Train Size: 80; training 20 times.....	Mean Error: 25.34% +- 1.76%
Train Size: 100; training 20 times.....	Mean Error: 19.79% +- 2.24%
Train Size: 120; training 20 times.....	Mean Error: 21.25% +- 4.89%



Example Bagging

```
ID3Inducer inducer("my_first_id3");
BaggingInd baggind("Baggind");
CatTestResult resultid3(inducer.get_categorizer(), train, "monk1",
    ".names", ".test");
Mcout << resultid3 << endl;
baggind.set_main_inducer(&inducer);
baggind.read_data("monk1");
baggind.train();
CatTestResult resultbagg(baggind.get_categorizer(), train,
    "monk1", ".names", ".test");
Mcout << resultbagg << endl;
```

Number of test instances: 432. Unseen: 308, seen 124.
Generalization error: 26.62%. Memorization error: 0.00%
Error: 18.98% +- 1.89% [15.56% - 22.95%]
Generalization error: 14.29%. Memorization error: 0.00%
Error: 10.19% +- 1.46% [7.67% - 13.40%]



Homework

- ☞ Read chapter 3 on decision tree learning
- ☞ Read at least one out the following three articles
 - ☞ "A study of cross-validation and bootstrap for accuracy estimation and model selection" [Kohavi 1995]
 - ☞ "Irrelevant features and the subset selection problem" [John, Kohavi, Pfleger]
 - ☞ "Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning" [Fayyad, Irani 1993]