Decision tree learning

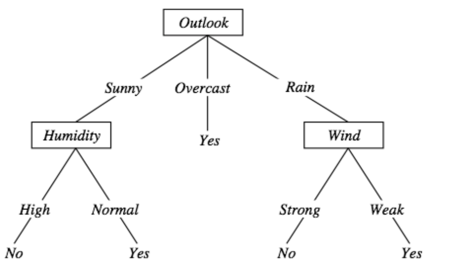
Maria Simi, 2010/2011

Machine Learning, Tom Mitchell
Mc Graw-Hill International Editions, 1997
(Cap 3).

Inductive inference with decision trees

- *Decision Trees* is one of the most widely used and practical methods of *inductive inference*
- Features
  - Method for approximating *discrete-valued* functions (including boolean)
  - Learned functions are represented as *decision trees* (or *if-then-else* rules)
  - Expressive hypotheses space, including disjunction
  - Robust to noisy data

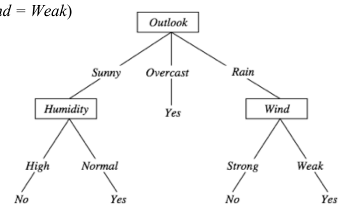
Decision tree representation (PlayTennis)



⟨*Outlook=Sunny, Temp=Hot, Humidity=High, Wind=Strong*⟩ No

Decision trees expressivity

- Decision trees represent a disjunction of conjunctions on constraints on the value of attributes:
  (*Outlook = Sunny* ∧ *Humidity = Normal*) ∨
  (*Outlook = Overcast*) ∨
  (*Outlook = Rain* ∧ *Wind = Weak*)



Decision trees representation

Learned from medical records of 1000 women

Negative examples are C-sections

```
[833+,167-] .83+ .17-
Fetal_Presentation = 1: [822+,116-] .88+ .12-
| Previous_Csection = 0: [767+,81-] .90+ .10-
| | Primiparous = 0: [399+,13-] .97+ .03-
| | Primiparous = 1: [368+,68-] .84+ .16-
| | | Fetal_Distress = 0: [334+,47-] .88+ .12-
| | | | Birth_Weight < 3349: [201+,10.6-] .95+ .05   +
| | | | Birth_Weight >= 3349: [133+,36.4-] .78+ .2   -
| | | Fetal_Distress = 1: [34+,21-] .62+ .38-
| Previous_Csection = 1: [55+,35-] .61+ .39-
Fetal_Presentation = 2: [3+,29-] .11+ .89-
Fetal_Presentation = 3: [8+,22-] .27+ .73-
```

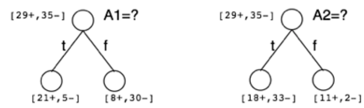

When to use Decision Trees

- Problem characteristics:
  - Instances can be described by attribute value pairs
  - Target function is discrete valued
  - Disjunctive hypothesis may be required
  - Possibly noisy training data samples
    - Robust to errors in training data
    - Missing attribute values
- Different classification problems:
  - Equipment or medical diagnosis
  - Credit risk analysis
  - Several tasks in natural language processing

Top-down induction of Decision Trees

- ID3 (Quinlan, 1986) is a basic algorithm for learning DT's
- Given a training set of examples, the algorithms for building DT performs search in the space of decision trees
- The construction of the tree is top-down. The algorithm is greedy.
- The fundamental question is "which attribute should be tested next? Which question gives us more information?"
- Select the *best* attribute
- A descendent node is then created for each possible value of this attribute and examples are partitioned according to this value
- The process is repeated for each successor node until all the examples are classified correctly or there are no attributes left

Which attribute is the best classifier?



- A statistical property called *information gain*, measures how well a given attribute separates the training examples
- Information gain uses the notion of *entropy*, commonly used in information theory
- *Information gain = expected reduction of entropy*

Entropy in binary classification

- Entropy measures the *impurity* of a collection of examples. It depends from the distribution of the random variable *p*.
  - S is a collection of training examples
  - p_+ the proportion of positive examples in S
  - p_- the proportion of negative examples in S

$Entropy\ (S) \equiv -p_+ \log_2 p_+ - p_- \log_2 p_-$ $\quad [0 \log_2 0 = 0]$

$Entropy\ ([14+, 0-]) = -14/14 \log_2 (14/14) - 0 \log_2 (0) = 0$

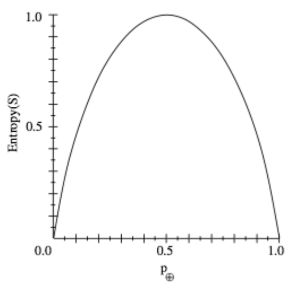
$Entropy\ ([9+, 5-]) = -9/14 \log_2 (9/14) - 5/14 \log_2 (5/14) = 0{,}94$

$Entropy\ ([7+, 7-]) = -7/14 \log_2 (7/14) - 7/14 \log_2 (7/14) =$

$= 1/2 + 1/2 = 1$ $\quad [\log_2 1/2 = -1]$

Note: the log of a number < 1 is negative, $0 \le p \le 1$, $0 \le entropy \le 1$

Entropy



Entropy in general

- Entropy measures the amount of information in a random variable

$H(X) = -p_+ \log_2 p_+ - p_- \log_2 p_-$ $\qquad X = \{+, -\}$

for binary classification [two-valued random variable]

$$H(X) = -\sum_{i=1}^{c} p_i \log_2 p_i = \sum_{i=1}^{c} p_i \log_2 1/p_i \qquad X = \{i, \ldots, c\}$$

for classification in c classes

Example: rolling a die with 8, equally probable, sides

$$H(X) = -\sum_{i=1}^{8} 1/8 \log_2 1/8 = -\log_2 1/8 = \log_2 8 = 3$$

Entropy and information theory

- Entropy specifies the number the average length (in bits) of the message needed to transmit the outcome of a random variable. This depends on the probability distribution.
- Optimal length code assigns $\lceil -\log_2 p \rceil$ bits to messages with probability *p.* Most probable messages get shorter codes.
- Example: 8-sided [unbalanced] die

1	2	3	4	5	6	7	8
4/16	4/16	2/16	2/16	1/16	1/16	1/16	1/16
2 bits	*2 bits*	*3 bits*	*3 bits*	*4bits*	*4bits*	*4bits*	*4bits*

$E = (1/4 \log_2 4) \times 2 + (1/8 \log_2 8) \times 2 + (1/16 \log_2 16) \times 4 = 1+3/4+1 = 2{,}75$

Information gain as *entropy reduction*

- *Information gain* is the *expected* reduction in entropy caused by partitioning the examples on an attribute.
- The higher the information gain the more effective the attribute in classifying training data.
- Expected reduction in entropy knowing *A*

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|Sv|}{|S|} Entropy(Sv)$$

$Values(A)$ possible values for A

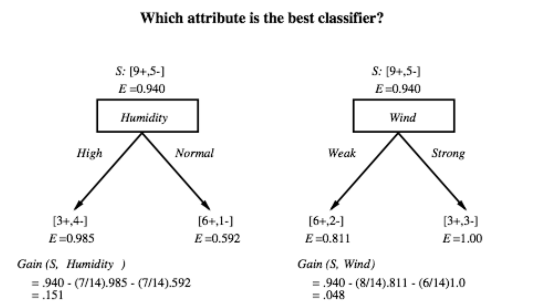
Sv subset of S for which A has value v

Example: expected information gain

- Let
  - $Values(Wind) = \{Weak, Strong\}$
  - $S = [9+, 5-]$
  - $S_{Weak} = [6+, 2-]$
  - $S_{Strong} = [3+, 3-]$
- Information gain due to knowing *Wind*:

$$Gain(S, Wind) = Entropy(S) - 8/14\ Entropy(S_{Weak}) - 6/14\ Entropy(S_{Strong})$$
$$= 0{,}94 - 8/14 \times 0{,}811 - 6/14 \times 1{,}00$$
$$= 0{,}048$$

Which attribute is the best classifier?



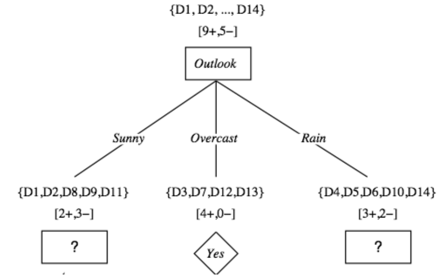
Example

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
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	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	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

First step: which attribute to test at the root?

- Which attribute should be tested at the root?
  - $Gain(S, Outlook) = 0.246$
  - $Gain(S, Humidity) = 0.151$
  - $Gain(S, Wind) = 0.084$
  - $Gain(S, Temperature) = 0.029$
- *Outlook* provides the best prediction for the target
- Lets grow the tree:
  - add to the tree a successor for each possible value of *Outlook*
  - partition the training samples according to the value of *Outlook*

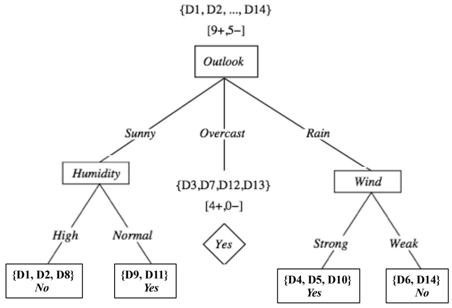
After first step



Second step

- Working on *Outlook=Sunny* node:
  - $Gain(S_{Sunny}, Humidity) = 0.970 - 3/5 \times 0.0 - 2/5 \times 0.0 = 0.970$
  - $Gain(S_{Sunny}, Wind) = 0.970 - 2/5 \times 1.0 - 3.5 \times 0.918 = 0 .019$
  - $Gain(S_{Sunny}, Temp.) = 0.970 - 2/5 \times 0.0 - 2/5 \times 1.0 - 1/5 \times 0.0 = 0.570$
- *Humidity* provides the best prediction for the target
- Lets grow the tree:
  - add to the tree a successor for each possible value of *Humidity*
  - partition the training samples according to the value of *Humidity*

Second and third steps

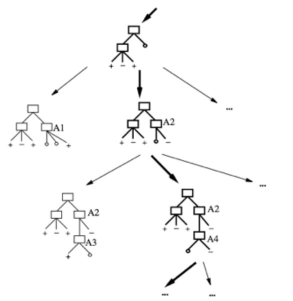


ID3: algorithm

ID3(*X, T, Attrs*) *X*: training examples:
T: target attribute (e.g. *PlayTennis*),
Attrs: other attributes, initially all attributes

Create *Root* node
If all X's are +, *return Root* with class +
If all X's are –, *return Root* with class –
If Attrs is empty *return Root* with class most common value of *T* in *X*
else
A ← best attribute; decision attribute for Root ← *A*
For each possible value v_i of *A:*
- add a new branch below *Root*, for test $A = v_i$
- X_i ← subset of *X* with $A = v_i$
- *If* X_i is empty *then* add a new leaf with class the most common value of *T* in *X*
else add the subtree generated by ID3(X_i, *T, Attrs* - {*A*})
return Root

Search space in Decision Tree learning



- The search space is made by partial decision trees
- The algorithm is *hill-climbing*
- The evaluation function is *information gain*
- The hypotheses space is complete (represents all discrete-valued functions)
- The search maintains a single current hypothesis
- No backtracking; no guarantee of optimality
- It uses all the available examples (not incremental)
- May terminate earlier, accepting noisy classes

Inductive bias in decision tree learning

- What is the inductive bias of DT learning?

1. *Shorter trees are preferred over longer trees*
   Not enough. This is the bias exhibited by a simple breadth first algorithm generating all DT's e selecting the shorter one
2. *Prefer trees that place high information gain attributes close to the root*

- *Note*: DT's are not limited in representing all possible functions

Two kinds of biases

- Preference or search biases (due to the search strategy)
  - ID3 searches a *complete* hypotheses space; the search strategy is *incomplete*
- Restriction or language biases (due to the set of hypotheses expressible or considered)
  - *Candidate-Elimination* searches an *incomplete* hypotheses space; the search strategy is *complete*
- A combination of biases in learning a linear combination of weighted features in board games.

Prefer shorter hypotheses: Occam's rasor

- Why prefer shorter hypotheses?
- Arguments in favor:
  - There are fewer short hypotheses than long ones
  - If a short hypothesis fits data unlikely to be a coincidence
  - Elegance and aesthetics
- Arguments against:
  - Not every short hypothesis is a reasonable one.
- Occam's razor: *"The simplest explanation is usually the best one."*
  - a principle usually (though incorrectly) attributed14th-century English logician and Franciscan friar, William of Ockham.
  - lex parsimoniae ("law of parsimony", "law of economy", or "law of succinctness")
  - The term razor refers to the act of *shaving away* unnecessary assumptions to get to the simplest explanation.

Issues in decision trees learning

- Overfitting
  - Reduced error pruning
  - Rule post-pruning
- Extensions
  - Continuous valued attributes
  - Alternative measures for selecting attributes
  - Handling training examples with missing attribute values
  - Handling attributes with different costs
  - Improving computational efficiency
  - Most of these improvements in C4.5 (Quinlan, 1993)

Overfitting: definition

- Building trees that "adapt too much" to the training examples may lead to "overfitting".
- Consider error of hypothesis h over
  - training data: $error_D(h)$ empirical error
  - entire distribution X of data: $error_X(h)$ expected error
- Hypothesis *h overfits* training data if there is an alternative hypothesis $h' \in H$ such that

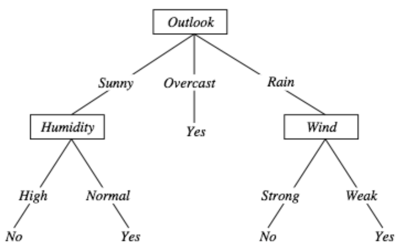
$$error_D(h) < error_D(h') \quad \text{and}$$
$$error_X(h') < error_X(h)$$

i.e. h' behaves better over unseen data

Example

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
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	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	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
D15	Sunny	Hot	Normal	Strong	No

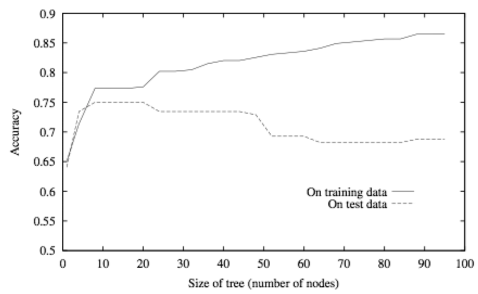
Overfitting in decision trees



⟨*Outlook=Sunny, Temp=Hot, Humidity=Normal, Wind=Strong, PlayTennis=No* ⟩

New noisy example causes splitting of second leaf node.

Overfitting in decision tree learning



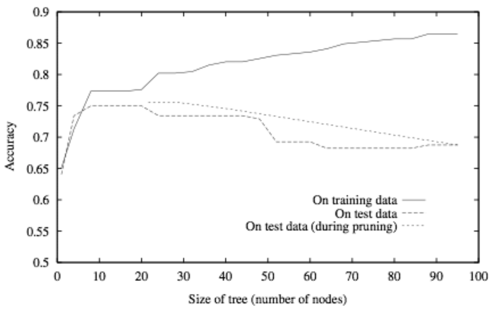
Avoid overfitting in Decision Trees

- Two strategies:
  1. Stop growing the tree earlier, before perfect classification
  2. Allow the tree to *overfit* the data, and then *post-prune* the tree
- Training and validation set
  - split the training in two parts (training and validation) and use validation to assess the utility of *post-pruning*
    - *Reduced error pruning*
    - *Rule pruning*
- Other approaches
  - Use a statistical test to estimate effect of expanding or pruning
  - *Minimum description length principle*: uses a measure of complexity of encoding the DT and the examples, and halt growing the tree when this encoding size is minimal

Reduced-error pruning (Quinlan 1987)

- Each node is a candidate for pruning
- *Pruning* consists in removing a subtree rooted in a node: the node becomes a leaf and is assigned the most common classification
- Nodes are removed only if the resulting tree performs no worse on the validation set.
- Nodes are pruned iteratively: at each iteration the node whose removal most increases accuracy on the validation set is pruned.
- Pruning stops when no pruning increases accuracy

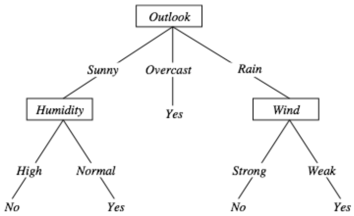
Effect of reduced error pruning



Rule post-pruning

1. Create the decision tree from the training set
2. Convert the tree into an equivalent set of rules
   - Each path corresponds to a rule
   - Each node along a path corresponds to a pre-condition
   - Each leaf classification to the post-condition
3. Prune (generalize) each rule by removing those preconditions whose removal improves accuracy …
   - … over validation set
   - … over training with a pessimistic, statistically inspired, measure
4. Sort the rules in estimated order of accuracy, and consider them in sequence when classifying new instances

Converting to rules



(*Outlook=Sunny*)∧(*Humidity=High*) ⇒ (*PlayTennis=No*)

Why converting to rules?

- Each distinct path produces a different rule: a condition removal may be based on a local (contextual) criterion. Node pruning is global and affects all the rules
- In rule form, tests are not ordered and there is no book-keeping involved when conditions (nodes) are removed
- Converting to rules improves readability for humans

Dealing with continuous-valued attributes

- So far discrete values for attributes and for outcome.
- Given a continuous-valued attribute A, dynamically create a new attribute A_c

  A_c = True *if* $A < c$, False *otherwise*
- How to determine threshold value c ?
- Example. *Temperature* in the *PlayTennis* example
  - Sort the examples according to *Temperature*

Temperature	40	48	| 60	72	80	| 90
PlayTennis	No	No	*54* Yes	Yes	Yes	*85* No

  - Determine candidate thresholds by averaging consecutive values where there is a change in classification: (48+60)/2=54 and (80+90)/2=85
  - Evaluate candidate thresholds (attributes) according to information gain. The best is $Temperature_{>54}$. The new attribute competes with the other ones

Problems with *information gain*

- Natural bias of information gain: it favours attributes with many possible values.
- Consider the attribute *Date* in the *PlayTennis* example.
  - *Date* would have the highest information gain since it perfectly separates the training data.
  - It would be selected at the root resulting in a very broad tree
  - Very good on the training, this tree would perform poorly in predicting unknown instances. Overfitting.
- The problem is that the partition is too specific, too many small classes are generated.
- We need to look at alternative measures …

An alternative measure: *gain ratio*

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

- S_i are the sets obtained by partitioning on value i of A
- *SplitInformation* measures the entropy of S with respect to the values of A. The more uniformly dispersed the data the higher it is.

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

- *GainRatio* penalizes attributes that split examples in many small classes such as *Date*. Let $|S|=n$, *Date* splits examples in n classes
  - $SplitInformation(S, Date)= -[(1/n \log_2 1/n)+\ldots+ (1/n \log_2 1/n)]= -\log_2 1/n = \log_2 n$
- Compare with A, which splits data in two even classes:
  - $SplitInformation(S, A)= - [(1/2 \log_2 1/2)+ (1/2 \log_2 1/2)]= - [- 1/2 -1/2]=1$

Adjusting *gain-ratio*

- Problem: $SplitInformation(S, A)$ can be zero or very small when $|S_i| \approx |S|$ for some value i
- To mitigate this effect, the following heuristics has been used:
  1. compute *Gain* for each attribute
  2. apply *GainRatio* only to attributes with *Gain* above average
- Other measures have been proposed:
  - *Distance-based* metric [Lopez-De Mantaras, 1991] on the partitions of data
  - Each partition (induced by an attribute) is evaluated according to the distance to the partition that perfectly classifies the data.
  - The partition closest to the *ideal* partition is chosen

Handling incomplete training data

- How to cope with the problem that the value of some attribute may be missing?
  - *Example*: Blood-Test-Result in a medical diagnosis problem
- The strategy: use other examples to guess attribute
  1. Assign the value that is most common among the training examples at the node
  2. Assign a probability to each value, based on frequencies, and assign values to missing attribute, according to this probability distribution
- Missing values in new instances to be classified are treated accordingly, and the most probable classification is chosen (C4.5)

Handling attributes with different costs

- Instance attributes may have an associated cost: we would prefer decision trees that use low-cost attributes
- ID3 can be modified to take into account costs:

1. Tan and Schlimmer (1990)

$$\frac{Gain^2(S, A)}{Cost(S, A)}$$

2. Nunez (1988)

$$\frac{2^{Gain(S, A)} - 1}{(Cost(A) + 1)^w} \quad w \in [0,1]$$

References

- Machine Learning, Tom Mitchell, Mc Graw-Hill International Editions, 1997 (Cap 3).