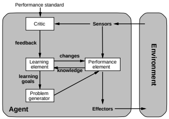
Clinical Decision Support Systems, 23/24

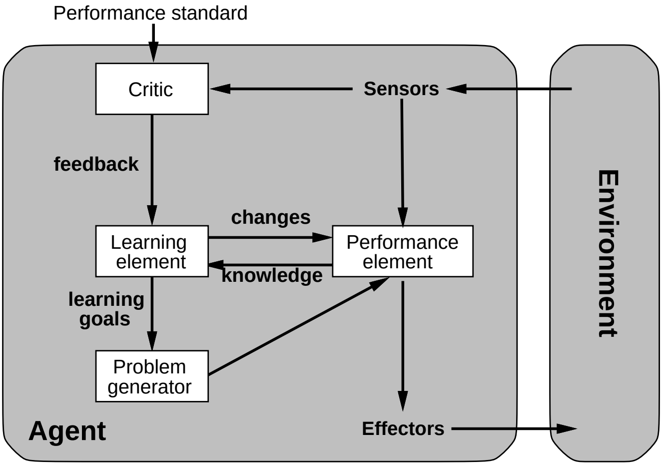
Inês Dutra and Pedro Rodrigues
DCC-FCUP & MEDCIDS-FMUP
ines@dcc.fc.up.pt, pprodrigues@med.up.pt

March 1st, 2024



Components of an agent that learns

(Fig. 2.15, AIMA book, 3rd ed., page 55)



Learning from observations

- Design of an intelligent system influenced by 4 factors:
  - identification of components to improve
  - representation used for data and components (logic?)
  - type of feedback
  - prior knowledge

Learning from observations

- Components that can be learned:
  - function that maps conditions of current state to actions
  - relevant properties of the environment (perception)
  - info about modifications of the environment
  - info about results of possible actions
  - info about utility of the results
  - info about action priorities
  - objectives that describe states that maximize utility

A definition for Learning

- "An agent **learns** if it improves its performance in future tasks after making observations about the past or current world." (Mitchel)

Machine Learning: very brief overview

- Learning?
  - Given observations O,
    described by features $f_1, f_2, \ldots, f_n$,
    the task of a machine learning algorithm is:
    - to find patterns based on features $f_1, f_2, \ldots, f_n$ (all or some of them), that distinguish among different groups of observations OR
    - to find a function that will **predict** new observations

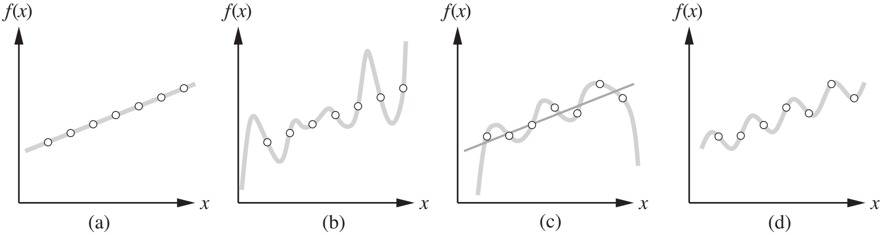
Machine Learning: very brief overview

- Learning?
  - Can be **supervised**:
    - Given features $f_1, f_2, \ldots, f_n$,
      and a special feature, the **target** variable (ground truth),
      find a model that can **predict** the target variable for **new** observations that are described by features $f_1, f_2, \ldots, f_n$
    - The supervised learning task can be **classification** or **regression**
  - Can be **unsupervised**: find subgroups of patterns, no target variable is known or provided
    - clustering
    - association rules
  - Other learning methods: reinforcement learning, matrix factorization for recommender systems
  - *background/prior knowledge*: description of observations, necessary to improve the learning

Inductive Learning

- In supervised learning, the learning element has a correct value or an approximate value estimated by a function over the inputs
- Learning will try to find a function that will approximate the true values of the variable being learned
- **Example**: pair $(x, f(x))$, where x is input and f(x) is the output (target variable)
- **Inductive inference** (or simply induction): given a set of observations f, returns a function h (**hypothesis**) that approximates f.
- **Bias**: preference for one hypothesis

Inductive Learning



Inductive Learning

- Alternative: **incremental learning**. Agent updates previous hypothesis for each new case instead of always inducing all
- Agent can also receive feedback about chosen actions
- Form in which hypotheses are represented: free
- Learning algorithms: various!
- At least two approaches to learn logical sentences: **decision trees** and **inductive logic programming** (more general, less efficient).
- Problem: how about representation of the function used to learn? Is it "representable" in the language? Is it efficient?

Inductive Learning

- First order logic: requires computational time and a good number of examples to learn a good set of sentences
- "Good" set of sentences: correctly predict future cases
- Problem: how to assess if a learning algorithm is producing a theory (hypothesis) that correctly predict future new unseen cases?

Decision Trees

- simple and easy to implement
- initially used for boolean decisions: yes/true or no/false
- Example: wait for a restaurant table
- Objective: learn a definition ("function") to "WillWait" represented as a decision tree

Decision Trees

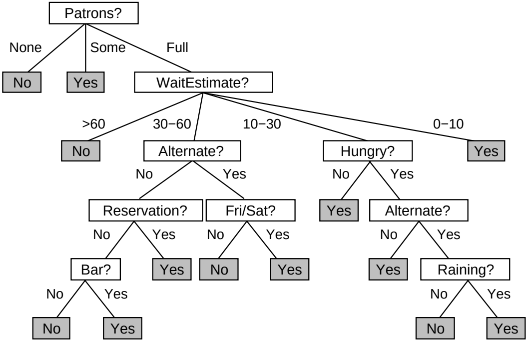
Observed variables (features, attributes):

- Alt: is there an alternative restaurant nearby?
- Bar: does the restaurant have a waiting area?
- Fri: true if it is Friday
- Hungry: am I hungry?
- Patrons: amount of people in the restaurant (None, Some, Full).
- Price: \$, \$\$, \$\$\$.
- Rain: is it raining?
- Reservation: do I have a reservation?
- Type: French, Italian etc.
- Estimated Waiting Time: 0–10min, 10–30, 30–60, $>$ 60.

Decision Trees

Ex	Attributes										Goal
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
X1	Yes	No	No	Yes	Some	$$$	No	Yes	French	0–10	Yes
X2	Yes	No	No	Yes	Full	$	No	No	Thai	30–60	No
X3	No	Yes	No	No	Some	$	No	No	Burger	0–10	Yes
X4	Yes	No	Yes	Yes	Full	$	No	No	Thai	10–30	Yes
X5	Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	No
X6	No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0–10	Yes
X7	No	Yes	No	No	None	$	Yes	No	Burger	0–10	No
X8	No	No	No	Yes	Some	$$	Yes	Yes	Thai	0–10	Yes
X9	No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	No
X10	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10–30	No
X11	No	No	No	No	None	$	No	No	Thai	0–10	No
X12	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30–60	Yes

Decision Tree for the restaurant example



Decision Trees

- In logic:
  $\forall\ r\ \ Pat(r, Full) \wedge WaitingTime(r, 10-30) \wedge \neg Hungry(r, N) \Rightarrow WillWait(r)$
- In its simplest form, decision trees can not represent tests over two or more different objects (every object needs to be "ground")
- Limitations in representation
- Any boolean function can be represented by a decision tree
- Representation of a decision tree must be compact, because truth-tables have exponential growth.

Decision Trees

- **Examples**: attribute values plus class value (feature vector).
- **Classification of an example**: predicted value of the class value for a given example.
- when value is true, example is **positive**, otherwise example is **negative**.
- full set of examples: **training set**.

Decision Trees

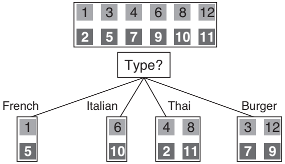
- How to induce a decision tree from examples?
- Each example can be a different path in the tree...
- ...but the classifier can not extract any pattern different from the ones used in the tree.
- To extract a pattern is to describe a large number of cases in a concise way.
- General principle of inductive learning: **Ockham's razor**. "The most probable hypothesis is the simplest consistent with all (or most) observations".
- To find a minimal decision tree is an intractable problem.
- Heuristics can help.

Decision Trees

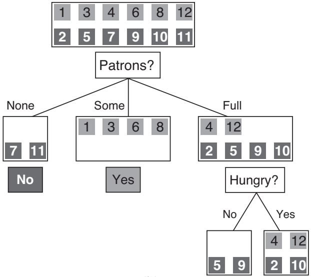
- Basic idea of the algorithm: test "most important" attributes first.
- What is a "most important" attribute?
- Example: 12 observations, separated in positive and negative sets.
- *Patrons* is an important attribute: if its value is None or Some, the predicate has always a definite value: No or Yes.
- *Type*: poor attribute.
- Algorithm chooses the strongest attribute and places it as the root of the subtree.

Decision Trees

Choice between two attributes: Type and Patrons. Patrons is chosen because it distinguishes better positive (willWait=Yes) and negative (willWait=No) examples.



(a)



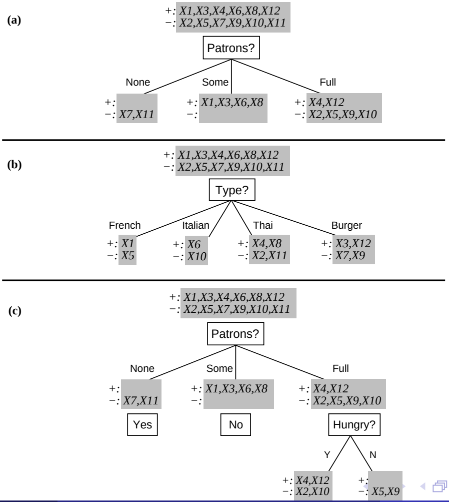
(b)

Decision Trees

- There are still subsets of examples not yet classified. The algorithm is recursively applied. There are 4 possible cases:
  - If there are still positive and negative examples to be classified, select the best attribute to split them.
  - If all remaining examples are positive (or negative), create a leaf to answer Yes (or No). Return.
  - If there are no more examples left, it means there is no observation in that path. Return Yes or No value depending on the majority class of the parent node.
  - If there are no more attributes left, but there are remaining examples, this means that those examples have exactly the same description, but different classifications. Simple solution: return majority class of these examples.

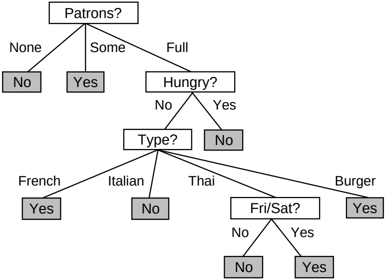
Decision Trees

Choice of attribute Patrons and continuation of the algorithm with the choice of the next best attribute: Hungry (c)



Decision Trees

Possible tree generated by an inductive decision tree learning algorithm.



Decision Trees

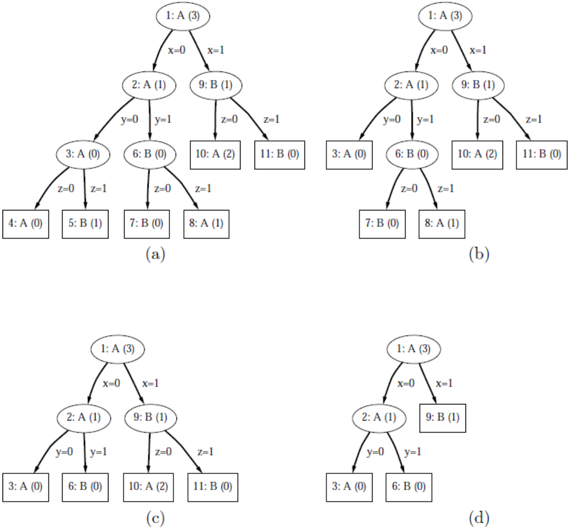
- Notes:
  - algorithm may conclude facts that are not evident from the examples. For example, always wait for a Thai restaurant if it is a weekend.
  - Because of this, precious amount of time can be wasted looking for bugs that do not exist.
  - The more examples, the most detailed will be the decision tree.
  - In this example, the tree can answer with an error, because it never saw a case where the waiting time is 0-10 minutes, but the restaurant is full
- Question: if the algorithm induces a consistent tree, but makes mistakes when classifying some examples, how incorrect is the tree?

Decision Trees

- Pruning consists in removing redundant nodes.
- The most common approach is to perform post-pruning.
- One of the simplest forms of post-pruning is reduced error pruning.
- Starting at the leaves, each node is replaced with its most popular class.
- If the prediction accuracy is not affected then the change is kept.
- While somewhat naive, reduced error pruning has the advantage of simplicity and speed.

Decision Trees

Example of pruning. (from Eibe Frank's PhD thesis Pruning Decision Trees and Lists)



Information Theory

- Used to find formal metrics to categorize attributes as "good" ou "reasonable" or "poor" etc.
- Information represented in number of bits.
  If I(p) = 1, we need 1 bit of information.
  If I(p) = 0, we do not need additional information.
- Let an attribute have n possible distinct values with probabilities $P(v_i), 1 \leq i \leq n$. Total information:

$$I(P(v_1), \ldots, P(v_n)) = \sum_{i=1}^{n} -P(v_i) log_2 P(v_i)$$

- Coding of the info with optimal size will have $log_2 p$ bits for an attribute with probability p.

Information Theory

- Considering positive and negative examples:

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n} log_2 \frac{p}{p+n} - \frac{n}{p+n} log_2 \frac{n}{p+n}$$

  is the estimator of the info contained in a correct answer.

- **Information Gain**: difference between the original information and the information after adding a new attribute:
  $Gain(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - Remaining(A)$
- Heuristic chooses attribute with higher gain (lower entropy).
- Ex: $Gain(Patrons) = 1 - [\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I(\frac{2}{6}, \frac{4}{6})] \approx 0.541$ bits.
- The "1" in the formula comes from the initial information: we have 6 positive examples (willWait=Yes) and 6 negative examples (willWait=no). Initial info: $-\frac{6}{12} log_2 \frac{6}{12} - \frac{6}{12} log_2 \frac{6}{12} = 1$

Algorithm ID3 for Decision Tree Induction

```
ID3(Examples, Target_Attribute, Attributes)
    Create a root node for the tree
    If all examples are positive,
        Return the single-node tree Root, with label = +.
    If all examples are negative,
        Return the single-node tree Root, with label = -.
    If number of predicting attributes is empty,
        Return the single node tree Root,
           with label = most common value of the
           target attribute in the examples.
    Else
        A = Attribute that best classifies examples
        Decision Tree attribute for Root = A
        For each possible value, vi, of A,
            Add a new tree branch below Root,
              corresponding to the test A = vi.
            Let Examples(vi) be the subset of examples that
              have the value vi for A
            If Examples(vi) is empty
               below this new branch add a leaf node with
                  label = most common target value in the examples
            Else
               below this new branch add the subtree
                  ID3 (Examples(vi), Target_Attribute, Attributes - {A})
            EndIf
        EndFor
    EndIf
    Return Root
```

ID3 algorithm

- Limitations:
  - information gain is useful only for problems with two classes
  - ID3 algorithm does not deal with numerical values
- Alternatives for attribute utility: jini index, gain ratio etc
- Alternative algorithms that handle numerical values: C4.5, C5.0, J48 (implementation of C4.5 in WEKA)
- When handling numerical values, discretization is needed.
- Methods: non-supervised (fixed width, fixed frequency or clustering) or supervised.
- Simple supervised method: 1Rule.
- 1Rule: works with the attribute and with the class variable. Sorts the attribute values and splits at each change of class. It is common to determine a minimum number of elements to place in an interval before splitting.

Information Theory

- Developing entropy calculation:

$$Entropy(Patrons) = [\frac{2}{12}(-\frac{0}{2}log_2\frac{0}{2} - \frac{2}{2}log_2\frac{2}{2})+$$

$$+\frac{4}{12}(-\frac{4}{4}log_2\frac{4}{4} - \frac{0}{4}log_2\frac{0}{4})+$$

$$+\frac{6}{12}(-\frac{2}{6}log_2\frac{2}{6} - \frac{4}{6}log_2\frac{4}{6})] \approx 0.46 bits.$$

One more example

Instance	Age	Type	Astigmatism	Tear production	Class
1	young	myope	no	reduced	none
2	young	myope	no	normal	soft
3	young	myope	yes	reduced	none
4	young	myope	yes	normal	hard
5	young	hypermetrope	no	reduced	none
6	young	hypermetrope	no	normal	soft
7	young	hypermetrope	yes	reduced	none
8	young	hypermetrope	yes	normal	hard
9	pre-presbyopic	myope	no	reduced	none
10	pre-presbyopic	myope	no	normal	soft
11	pre-presbyopic	myope	yes	reduced	none
12	pre-presbyopic	myope	yes	normal	hard
13	pre-presbyopic	hypermetrope	no	reduced	none
14	pre-presbyopic	hypermetrope	no	normal	soft
15	pre-presbyopic	hypermetrope	yes	reduced	none
16	pre-presbyopic	hypermetrope	yes	normal	none
17	presbyopic	myope	no	reduced	none
18	presbyopic	myope	no	normal	none
19	presbyopic	myope	yes	reduced	none
20	presbyopic	myope	yes	normal	hard
21	presbyopic	hypermetrope	no	reduced	none
22	presbyopic	hypermetrope	no	normal	soft
23	presbyopic	hypermetrope	yes	reduced	none
24	presbyopic	hypermetrope	yes	normal	none

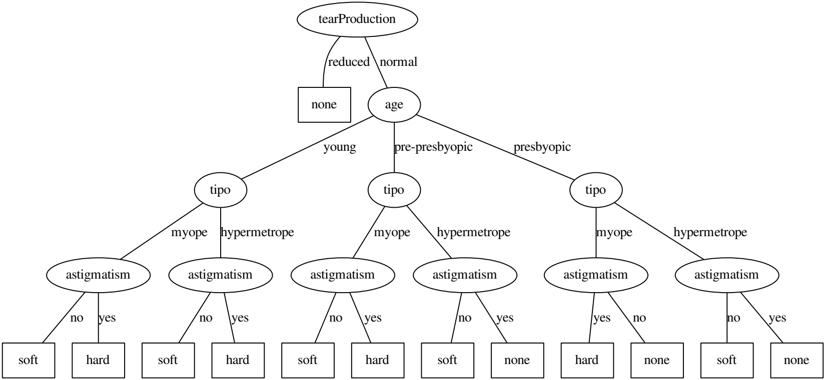
Table: Features of patients. Task: prescription of contact lenses

Decision Trees: exercise

- Given the table of the previous slide, produce a decision tree using all available variables (age, type, astigmatism, tear production and class) that can predict what kind of lenses a patient must use: hard, soft or none.
- What variable is the most relevant to distinguish among the class values?

Decision Trees: exercise

Possible tree (built manually):



Decision Trees: exercise

Possible tree (more compact, generated by software):