Sapienza University of Rome

Master in Artificial Intelligence and Robotics
Master in Engineering in Computer Science

Machine Learning

A.Y. 2020/2021

Prof. L. Iocchi, F. Patrizi

Sapienza University of Rome, Italy - Machine Learning (2020/2021)

Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree

3. Decision Trees

L. Iocchi, F. Patrizi

These learning methods are among the most popular of inductive inference algorithms and have been successfully applied to a broad range of tasks from Learning to diagnose medical cases to learning to assess credit risk of loan applicants

Overview

- Decision tree representation
- ID3 learning algorithm
- Entropy, Information gain
- Inductive Bias
- Overfitting and pruning

References

T. Mitchell. Machine Learning. Chapter 3

Concept Learning as search

Problem: Given a training set D for a target function c, compute the "best" consistent hypothesis wrt D.

Solution approach

- Define hypothesis space H
- Implement an algorithm to find $h^* \in H$, "best" hypothesis consistent with D

Decision Trees

even if it can be applied to continuous problems int he first part of this lecture we focus on the discree representation of the problem

Consider a discrete input space described with m attributes
$X = A_1 \times \ldots \times A_m$, with A_i finite
and a classification problem for $f : X \to C$

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.

Hypothesis space H: set of decision trees.

Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute

Decision tree

Tree with the following characteristics:

- Each internal node tests an attribute A_i
- Each branch denotes a value of an attribute $a_{i,j} \in A_i$
- Each leaf node assigns a classification value $c \in C$

Example *PlayTennis*

Instances $X = Outlook \times Temperature \times Humidity \times Wind$

$$Outlook = \{Sunny, Overcast, Rain\}$$

$$Temperature = \{Hot, Mild, Cold\}$$

$$Humidity = \{Normal, High\}$$

$$Wind = \{Weak, Strong\}$$

Classification values:

$$PlayTennis = \{Yes, No\}$$

Learning problem:

$$f : Outlook \times Temperature \times Humidity \times Wind \to PlayTennis$$

Example *PlayTennis*: Training data

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

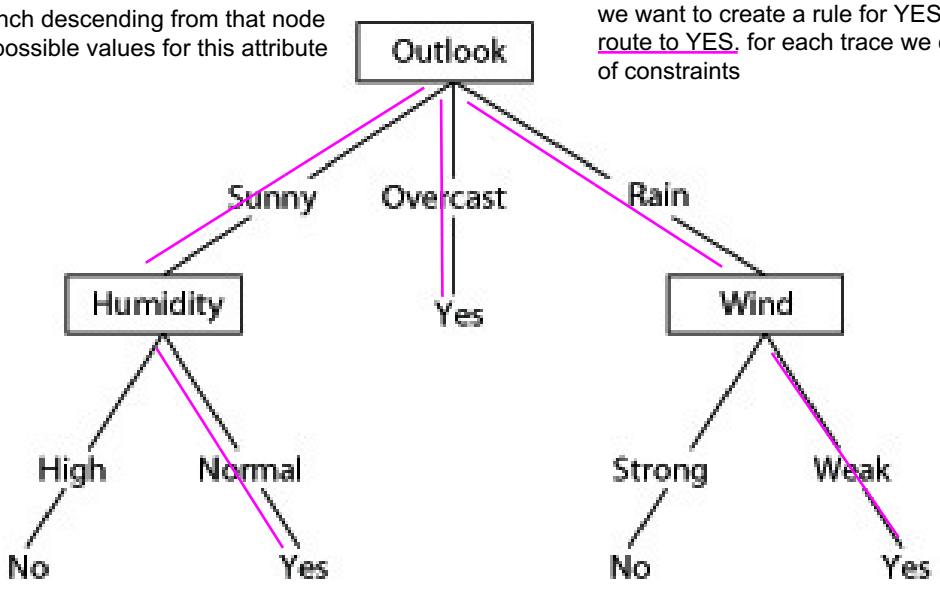
Decision Tree for *PlayTennis*

if we see the tree, we can find a path for labeling a new instance

it is one of the best method because it provide a good explaination of why we have a specific solution

Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute

we want to create a rule for YES, so we consider all the route to YES. for each trace we consider the conjunction of constraints



Decision Tree for *PlayTennis*

In general, decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances. Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions.

Decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances.

$$(Outlook = Sunny \wedge Humidity = Normal) \quad \vee$$
$$(Outlook = Overcast) \quad \vee$$
$$(Outlook = Rain \wedge Wind = Weak)$$

Disjunction of conjunctions of all the paths to positive (true) leaf nodes.

Converting a Tree to Rules

A rule is generated for each path to a leaf node. set of IF - THEN statements

IF $(Outlook = Sunny) \wedge (Humidity = High)$
THEN $PlayTennis = No$

IF $(Outlook = Sunny) \wedge (Humidity = Normal)$
THEN $PlayTennis = Yes$

...

Decisions are made explicit.

decision tree learning is generally best suited to problems with the following characteristics:
- Instances are represented by attribute-value pairs. Instances are described by a fixed set of attributes (e.g., Temperature) and their values (e.g., Hot)
- The target function has discrete output values. (e.g. yes or no)
- Disjunctive descriptions may be required
- The training data may contain errors.
- The training data may contain missing attribute values.

This approachis exemplifiedby the ID3 algorithm (Quinlan 1986) and its successor C4.5 (Quinlan 1993)

ID3 Algorithm

1) +
2) -
3) +/- majority sample in D

3 can be both + and - but the list of attributes is empty so we have just used all attributes in other part of the tree

codiminio

Input: *Examples*, *Target_attribute*, *Attributes*

Output: *Decision Tree* the best DT according to some matrics that we have seen

ID3, learns decision trees by constructing them topdown, beginning with the question "which attribute should be tested at the root of the tree?" To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples. The best attribute is selected and used as the test at the root node of the tree. A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node

1. Create a *Root* node for the tree
2. **if** all *Examples* are positive, **then** return the node *Root* with label $+$
3. **if** all *Examples* are negative, **then** return the node *Root* with label $-$
4. **if** *Attributes* is empty, **then** return the node *Root* with label $=$ most common value of *Target_attribute* in *Examples*
5. Otherwise ...

ID3 Algorithm

5. Otherwise
   - $A \leftarrow$ the "best" decision attribute for *Examples*
   - Assign A as decision attribute for *Root*
   - For each value v_i of A
     - add a new branch from *Root* corresponding to the test $A = v_i$
     - $Examples_{v_i}$ = subset of *Examples* that have value v_i for A
     - **if** $Examples_{v_i}$ is empty **then** add a leaf node with label = most common value of *Target_attribute* in *Examples*
     - **else** add the tree ID3($Examples_{v_i}$, *Target_attribute*, *Attributes*$-\{A\}$)

Which is the best attribute to choose?

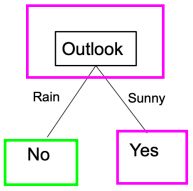
Output tree depends on attribute order

we have to choose the right attribute to start from

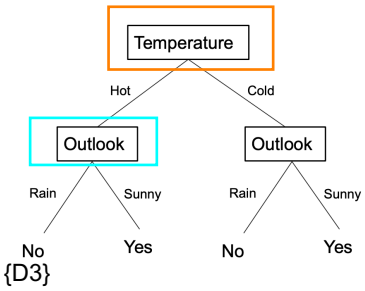
we have to choose an attribute that is able to better split the examples according to the value of this attribute in such a way that all the subsets are more strongly convinced between for YES and NO

Day	Outlook	Temperature	PlayTennis
D1	Sunny	Hot	Yes
D2	Sunny	Cold	Yes
D3	Rain	Hot	No
D4	Rain	Cold	No

Outlook first:



Temperature first:



Which is the best attribute to choose?

The central choice in the ID3 algorithm is selecting which attribute to test at each node in the tree. We would like to select the attribute that is most useful for classifying examples

Information gain measures how well a given attribute *separates* the training examples according to their target classification.

statistical property

ID3 uses this information gain measure to select among the candidate attributes at each step while growing the tree

Day	Outlook	Temperature	PlayTennis
D1	Sunny	Hot	Yes
D2	Sunny	Cold	Yes
D3	Rain	Hot	No
D4	Rain	Cold	No

Day	Outlook	Temperature	PlayTennis
D1	**Sunny**	Hot	**Yes**
D2	**Sunny**	Cold	**Yes**
D3	**Rain**	Hot	**No**
D4	**Rain**	Cold	**No**

Day	Outlook	Temperature	PlayTennis
D1	Sunny	**Hot**	**Yes**
D3	Rain	**Hot**	**No**
D2	Sunny	**Cold**	**Yes**
D4	Rain	**Cold**	**No**

ID3 selects the attribute that induces **highest information gain**.

In order to defineinformation gain precisely, we begin by defining a measure commonly used in information theory, called entropy, that characterizes the (im)purity of an arbitrary collection of examples

Information gain measured as reduction in *entropy*.

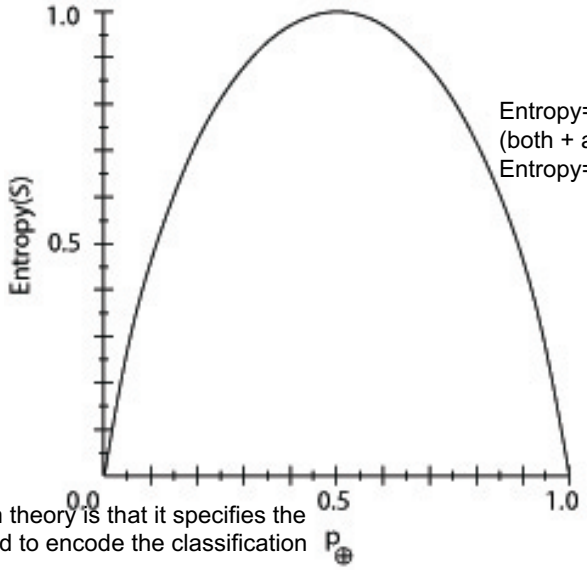
Entropy

given a population that comes as a set of examples with binary labels, so we have a proportional positive sampes and negative samples, the etropy of this set is given by

- $p_\oplus$ is the proportion of positive examples in S
- $p_\ominus$ $(= 1 - p_\oplus)$ is the proportion of negative examples in S
- Entropy measures the *impurity* of S

$$Entropy(S) \equiv -p_\oplus \log_2 p_\oplus - p_\ominus \log_2 p_\ominus$$

the goal is to reduce the entropy. So we should choose the attribute that when applied to partition the dataset according to values of this attribute, the entropy reduces. so in the previous examples, taking attribute Outlook the entropy=0, while choosing the Temperature entropy=0.5, because the outpu is 0.5 YES and 0.5 NO



Entropy=0.5 means that we cannot take a decision (both + and -)
Entropy=0 means that you can take a decision

Note the entropy is 1 when the collection contains an equal number of positive and negative examples. If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1

One interpretation of entropy from information theory is that it specifies the minimum number of bits of information needed to encode the classification of an arbitrary member of S

Entropy example

Thus far we have discussed entropy in the special case where the target classification is boolean. More generally, if the target attribute can take on c different values, then the entropy of S relative to this c-wise classification is defined as

Consider the set $S = [9+, 5-]$ (9 positive examples, 5 negative examples)

$$Entropy(S) = -(9/14)log_2(9/14) - (5/14)log_2(5/14) = 0.940$$

Note: we define $0 log_2 0 = 0$.

Maximum entropy when $p_\oplus = 0.5$, minimum when $p_\oplus = 0$ or 1.
In case of multi-valued target functions (c-wise classification)

extended to many classes

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i$$

proportion of S belonging to class i

N.B: Note the logarithm is still base 2 because entropy is a measure of the expected encoding length measured in bits.
Note also that if the target attribute can take on c possible values, the entropy can be as large as log2(c)

given a data set S and an attribute A, the Information Gain measures the reduction in entropy when we split the dataset according to the value of A

Information Gain

Given entropy as a measure of the impurity in a collection of training examples, we can now define a measure of the effectiveness of an attribute in classifying the training data

is simply the expected reduction in entropy caused by partitioning the examples according to this attribute

$Gain(S, A)$ = expected reduction in entropy of S caused by knowing the value of attribute A.

weighted average of the entropy over S

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

$Values(A)$: set of all possible values of A
$S_v = \{s \in S | A(s) = v\}$

Information Gain example

$$\begin{aligned} Values(Wind) &= \{Weak, Strong\} \\ S &= [9+, 5-] \\ S_{Weak} &= [6+, 2-] \\ S_{Strong} &= [3+, 3-] \end{aligned}$$

$$\begin{aligned} Gain(S, Wind) &= Entropy(S) - \frac{8}{14} Entropy(S_{Weak}) - \frac{6}{14} Entropy(S_{Strong}) \\ &= 0.940 - \frac{8}{14} 0.811 - \frac{6}{14} 1.00 \\ &= 0.048 \end{aligned}$$

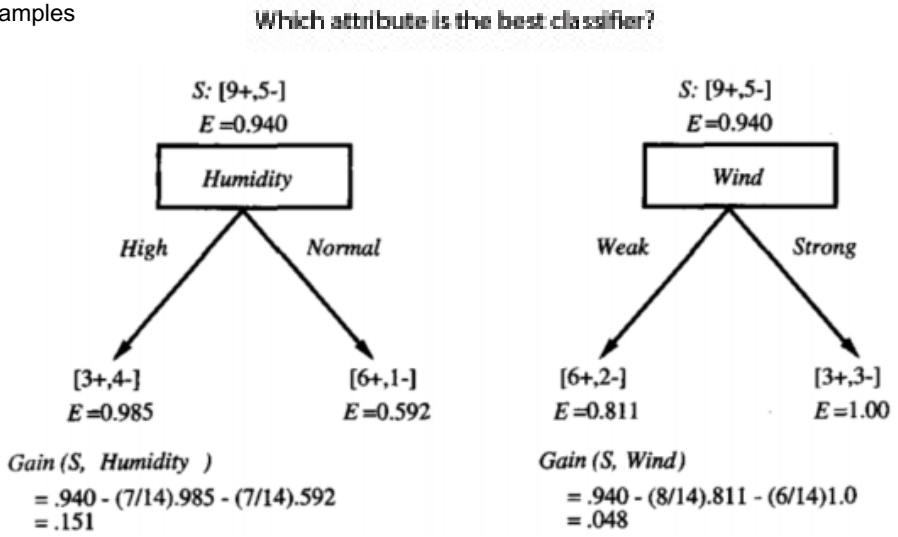
-(9/14*log(9/14)+5/14*log(5/14))

-(6/8*log(6/8)+2/8*log(2/8))

-(3/6*log(3/6)+3/6*log(3/6))

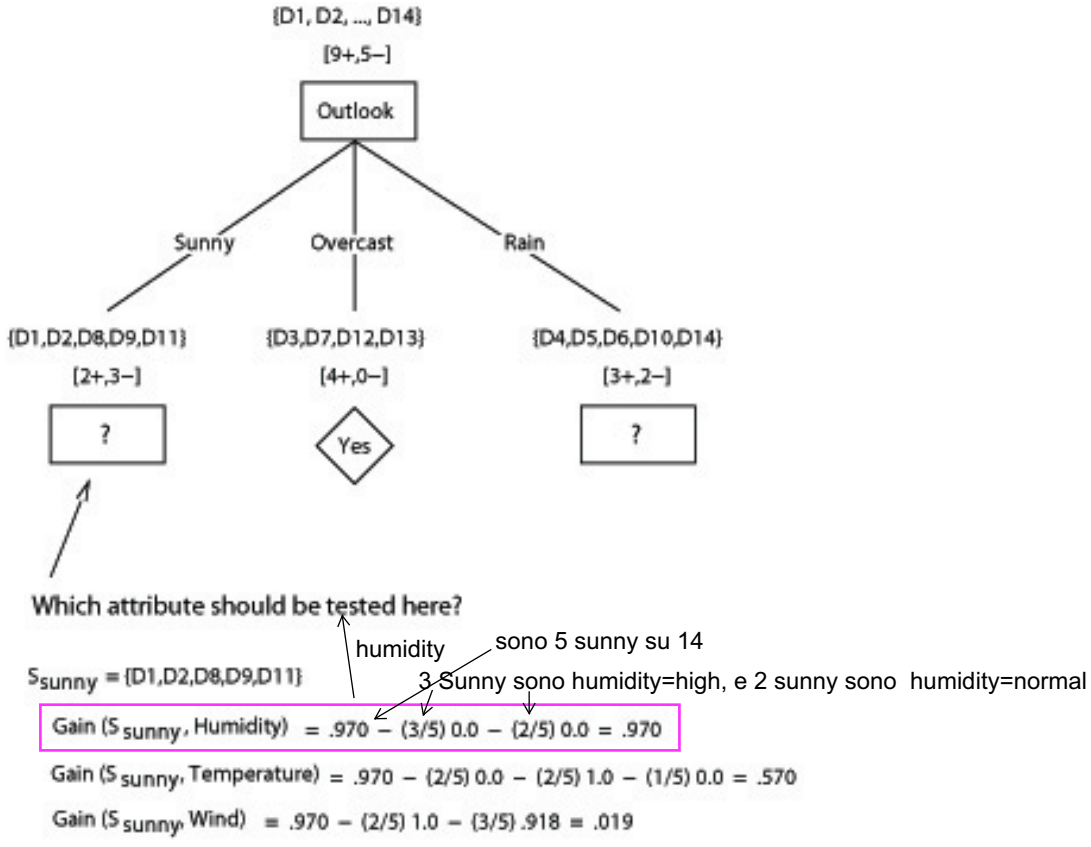
Selecting the Next Attribute

In this figure the information gain of two different attributes, Humidity and Wind,is computed in order to determine which is the better attribute for classifying the training examples



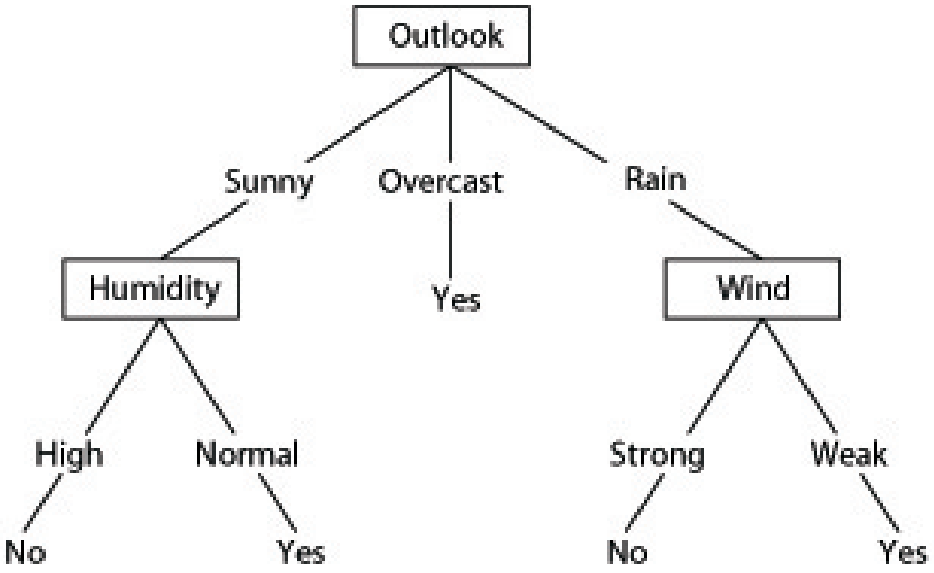
$$\text{Gain}(S,\text{Outlook}) = 0.246$$
$$\text{Gain}(S,\text{Humidity}) = 0.151$$
$$\text{Gain}(S,\text{Wind}) = 0.048$$
$$\text{Gain}(S,\text{Temperature}) = 0.029$$

Selecting the Next Attribute



Decision Tree for *PlayTennis*

attributes with highest Information Gain are close to the top levels

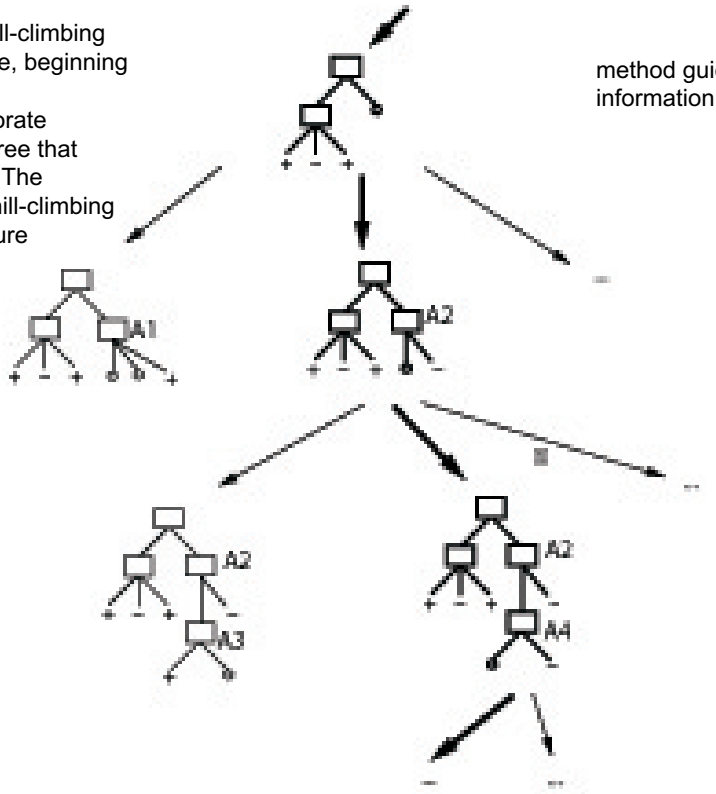


Hypothesis Space Search by ID3

like in AI

The hypothesis space searched by ID3 is the set of possible decision trees.

ID3 performs a simple-tocomplex, hill-climbing search through this hypothesis space, beginning with the empty tree, then consideringprogressively more elaborate hypotheses in search of a decision tree that correctly classifies the training data. The evaluation function that guides this hill-climbing search is the information gain measure



method guided by the heuristic with information gain criterion

Hypothesis Space Search by ID3

hp space is a complete space of finite discrete-valued functions, relative to the available attributes Because every finite discrete-valued function can be representedby some decision tree, ID3 avoids that the hypothesis space might not contain the target function.

- Hypothesis space is complete (target concept is there!)
- Outputs a single hypothesis (cannot determine how many DTs are consistent!) ID3 maintains only a single current hypothesis as it searches through the space of decision trees. By determining only a single hypothesis, ID3 loses the capabilities that follow from explicitly representing all consistent hypotheses
- No back tracking (local minima!) ID3 in its pure form performs no backtracking in its search
- Statistically-based search choices (robust to noisy data!)
- Uses all the training examples at each step (not incremental!)

ID3 uses all training examples at each step in the search to make statistically based decisions regarding how to refine its current hypothesis. This contrasts with methods that make decisions incrementally, based on individual training examples

Issues in Decision Tree Learning

- Determining how deeply to grow the DT
- Handling continuous attributes
- Choosing appropriate attribute selection measures
- Handling training data with missing attribute values
- Handling attributes with different costs

Overfitting in Decision Trees

this simple algorithm can produce trees that overfit the training examples

Consider a new data set $D' = D \cup d_{15}$ adding a training example:

$$d_{15} = \langle Sunny,\ Hot,\ Normal,\ Strong,\ PlayTennis = No \rangle$$

ID3 will generate a different tree T'

Note: T is consistent with D and T' is consistent with D' (i.e., accuracy = 100%)

Is T' in general a better solution for our learning problem?

We will say that a hypothesis overfits the training examples if some other hypothesisthat fits the training examples less well actually performs better over the entire distribution of instances

Definition: Given a hypothesis space H, a hypothesis $h \in H$ is said to **overfit** the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.

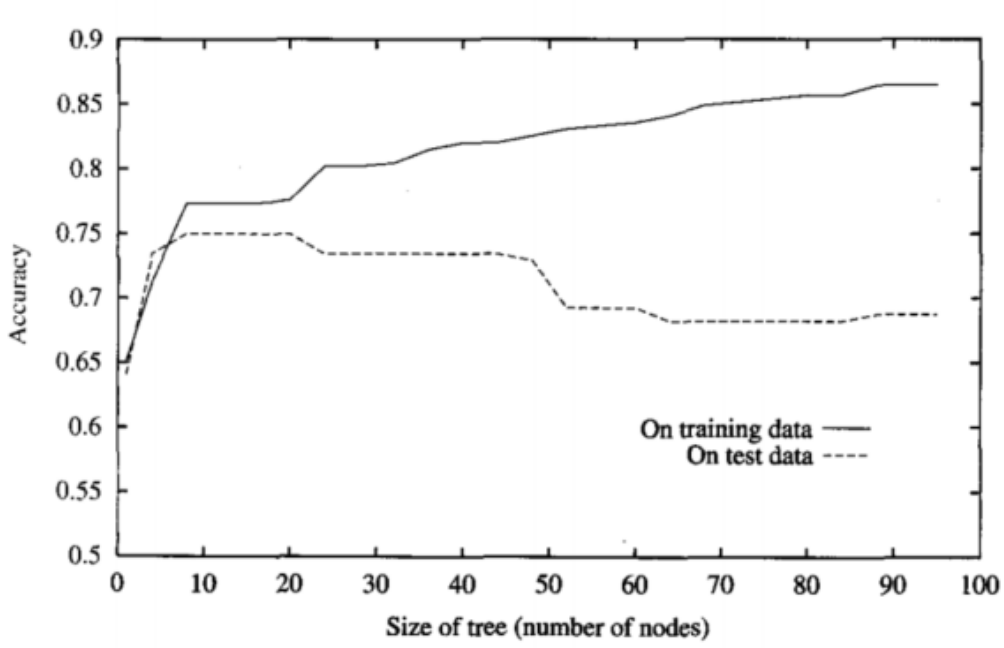
Overfitting in Decision Trees

Let's consider a more complex problem with $|D| \gg 15$ and containing noisy data and two decision trees T and T' obtained with different configuration of an ID3-like algorithm.

$$accuracy_D(T') > accuracy_D(T)$$ in the dataset

Is T' in general a better solution for our learning problem?

Overfitting in Decision Tree Learning



Avoiding Overfitting

How can we avoid overfitting?

approaches that stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data,

- stop growing when data split not statistically significant
- grow full tree, then post-prune — approaches that allow the tree to overfit the data, and then post-prune the tree.

To determine the correct tree size

- use a separate set of examples (distinct from the training examples) to evaluate the utility of post-pruning — cutting parts of the tree until you increase performance on the test data
- apply a statistical test to estimate accuracy of a tree on the entire data distribution — Use all the available data for training, but apply a statistical test to estimate whether expanding (or pruning) a particular node is likely to produce an improvement beyond the training set
- using an explicit measure of the complexity for encoding the examples and the decision trees.

The first of the above approaches is the most common and is often referred to as a training and validation set approach. In this approach, the available data are separated into two sets of examples: a training set, which is used to form the learned hypothesis, and a separate validation set, which is used to evaluate the accuracy of this hypothesis over subsequent data and, in particular, to evaluate the impact of pruning this hypothesis

Reduced-Error Pruning

How exactly might we use a validation set to prevent overfitting? One approach, called reduced-error pruning (Quinlan 1987), is to consider each of the decision nodes in the.tree to be candidates for pruning.

Pruning a decision node consists of removing the subtree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node. Nodes are removed only if the resulting pruned tree performs no worse than-the original over the validation set

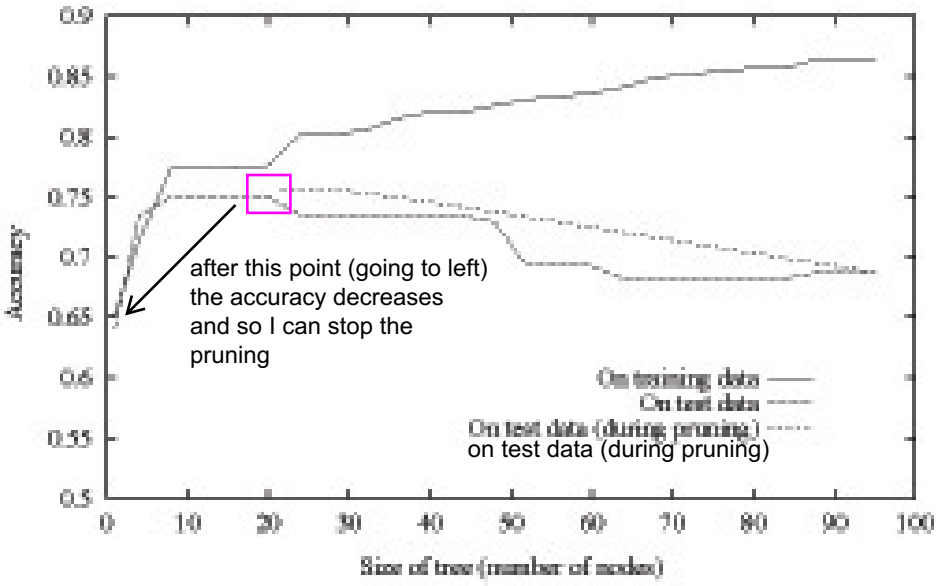
Split data into *training* and *validation* set implemented in 2 ways

Do until further pruning is harmful (decreases accuracy):

1. Evaluate impact on *validation* set of pruning each possible node (remove all the subtree and assign the most common classification)
2. Greedily remove the one that most improves *validation* set accuracy

Effect of Reduced-Error Pruning

Using a separate set of data to guide pruning is an effective approach provided a large amount of data is available. The major drawback of this approach is that when data is limited, withholding part of it for the validation set reduces even further the number of examples available for training.



Remarks on Reduced-Error Pruning

- it produces smallest version of most accurate subtree (removing sub-trees added due to coincidental irregularities).
- When data set is limited, reducing the set of training examples (used as validation examples) can give bad results.

Rule Post-Pruning (C4.5)

one quite successful method for finding high accuracy hypotheses is a technique we shall call rule post pruning. A variant of this pruning method is used by C4.5 (Quinlan 1993)

we work on the rules that are generated from the tree

1. Infer the decision tree allowing for overfitting
2. Convert the learned tree into an equivalent set of rules
3. Prune (generalize) each rule independently of others
4. Sort final rules into desired sequence for use

Continuous Valued Attributes

Create a discrete attribute to test continuous variables

- $Temperature = 82.5$
- $(Temperature > 72.3) = t, f$

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

Attributes with Many Values

One way to avoid this difficulty is to select decision attributesbased on some measure other than information gain. One alternative measure that has been used successfully is the gain ratio (Quinlan 1986)

Problem:

- If attribute has many values, *Gain* will select it
- Imagine using $Date = Jun\_3\_1996$ as attribute

One approach: use *GainRatio* instead (other measure over the Info.gain)

The gain ratio measure penalizes attributes such as Date by incorporating a term, called split informution, that is sensitive to how broadly and uniformly the attribute splits the data

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

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

c subsets of examples resulting from partitioning S by the c-valued attribute A

where S_i is subset of S for which A has value v_i

Attributes with Costs

ID3 can be modified to take into account attribute costs by introducinga cost term into the attribute selection measure

Consider

- medical diagnosis, *BloodTest* has cost \$150
- robotics, *Width_from_1ft* has cost 23 sec.

How to learn a consistent tree with low expected cost?
Replace gain by

- Tan and Schlimmer (1990)

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

- Nunez (1988) ($w \in [0, 1]$ determines importance of cost)

$$\frac{2^{Gain(S,A)} - 1}{(Cost(A) + 1)^w}$$

Unknown Attribute Values

incomplete datasets so datasets for which for each sample we have not all the info (for ex. I don't remember if a day was rainy or sunny)

In certain cases, the available data may be missing values for some attributes. For example, in a medical domain in which we wish to predict patient outcome based on various laboratory tests, it may be that the lab test Blood-Test-Result is available only for a subset of the patients.

What if 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 same fashion

These probabilities can be estimated again based on the observed frequencies of the various values for A among the examples at node n

Other algorithms based on Decision Trees

forest: set of trees

different trees with variations (different split, different dataset...)

Random Forest: ensemble method that generates a set of decision trees with some random criteria and integrates their values into a final result.

Random criteria: 1) random subsets of data (bagging), 2) random subset of attributes (feature selection), ...

Integration of results: majority vote (most common class returned by all the trees).

Random Forests are less sensitive to overfitting.

Summary

- Decision Trees can represent classification function by **making decisions explicit**
- Learning as search in the hypothesis space with heuristics based on information gain
- Statistical method (some robustness to noisy data)
- Overfitting and pruning
- Used as basis of randomized ensemble methods