Inductive Learning

Decision Tree Method

(If it's not simple,
it's not worth learning it)

R&N: Chap. 18, Sect. 18.1–3

Much of this taken from slides of: Jean-Claude Latombe, Stanford University; Stuart Russell, UC Berkley; Lise Getoor, University of Maryland.

Motivation

- An AI agent operating in a complex world requires an awful lot of knowledge: state representations, state axioms, constraints, action descriptions, heuristics, probabilities, ...
- More and more, AI agents are designed to acquire knowledge through learning

What is Learning?

- Mostly generalization from experience:

  "Our experience of the world is specific, yet we are able to formulate general theories that account for the past and predict the future"
  M.R. Genesereth and N.J. Nilsson, in *Logical Foundations of AI*, 1987
- → Concepts, heuristics, policies
- Supervised vs. un-supervised learning

Contents

- Introduction to inductive learning
- Logic-based inductive learning:
  - Decision-tree induction

Logic-Based Inductive Learning

- Background knowledge KB
- Training set D (observed knowledge) that is not logically implied by KB
- **Inductive inference**: Find h such that KB and h imply D

h = D is a trivial, but un-interesting solution (data caching)

Rewarded Card Example

- Deck of cards, with each card designated by [r,s], its rank and suit, and some cards "rewarded"
- Background knowledge KB:
  $((r=1) \vee \ldots \vee (r=10)) \Leftrightarrow NUM(r)$
  $((r=J) \vee (r=Q) \vee (r=K)) \Leftrightarrow FACE(r)$
  $((s=S) \vee (s=C)) \Leftrightarrow BLACK(s)$
  $((s=D) \vee (s=H)) \Leftrightarrow RED(s)$
- Training set D:
  $REWARD([4,C]) \wedge REWARD([7,C]) \wedge REWARD([2,S]) \wedge \neg REWARD([5,H]) \wedge \neg REWARD([J,S])$

Rewarded Card Example

- Deck of cards, with each card designated by [r,s], its rank and suit, and some cards "rewarded"
- Background knowledge KB:
  ((r=1) v … v (r=10)) ⇔ NUM(r)
  ((r=J) v (r=Q) v (r=K)) ⇔ FACE(r)
  ((s=S) v (s=C)) ⇔ BLACK(s)
  ((s=D) v (s=H)) ⇔ RED(s)
- Training set D:
  REWARD([4,C]) ∧ REWARD([7,C]) ∧ REWARD([2,S]) ∧ ¬REWARD([5,H]) ∧ ¬REWARD([J,S])
- Possible inductive hypothesis:
  h ≡ (NUM(r) ∧ BLACK(s) ⇔ REWARD([r,s]))

There are several possible inductive hypotheses

Learning a Predicate
(Concept Classifier)

- Set E of objects (e.g., cards)
- Goal predicate CONCEPT(x), where x is an object in E, that takes the value True or False (e.g., REWARD)

Learning a Predicate
(Concept Classifier)

- Set E of objects (e.g., cards)
- Goal predicate CONCEPT(x), where x is an object in E, that takes the value True or False (e.g., REWARD)
- Observable predicates A(x), B(X), … (e.g., NUM, RED)
- Training set: values of CONCEPT for some combinations of values of the observable predicates

Example of Training Set

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

Example of Training Set

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	[illegible]	[illegible]	High	Weak	Yes
D4	[illegible]	[illegible]	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	Yes
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Ternary attributes

Goal predicate is PLAY-TENNIS

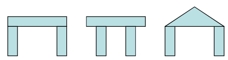
Note that the training set does not say whether an observable predicate is pertinent or not

Learning a Predicate
(Concept Classifier)

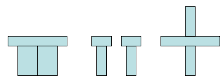
- Set E of objects (e.g., cards)
- Goal predicate CONCEPT(x), where x is an object in E, that takes the value True or False (e.g., REWARD)
- Observable predicates A(x), B(X), … (e.g., NUM, RED)
- Training set: values of CONCEPT for some combinations of values of the observable predicates
- Find a representation of CONCEPT in the form:
  CONCEPT(x) ⇔ S(A,B, …)
  where S(A,B,…) is a sentence built with the observable predicates, e.g.:
  CONCEPT(x) ⇔ A(x) ∧ (¬B(x) v C(x))

Learning an Arch Classifier

- These objects are arches: (positive examples)



- These aren't: (negative examples)



ARCH(x) ⇔ HAS-PART(x,b1) ∧ HAS-PART(x,b2) ∧
HAS-PART(x,b3) ∧ IS-A(b1,BRICK) ∧
IS-A(b2,BRICK) ∧ ¬MEET(b1,b2) ∧
(IS-A(b3,BRICK) v IS-A(b3,WEDGE)) ∧
SUPPORTED(b3,b1) ∧ SUPPORTED(b3,b2)

Example set

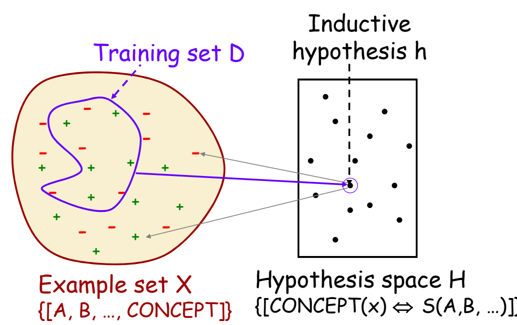
- An example consists of the values of CONCEPT and the observable predicates for some object x
- A example is positive if CONCEPT is True, else it is negative
- The set X of all examples is the example set
- The training set is a subset of X

a small one!

Hypothesis Space

- An hypothesis is any sentence of the form:
  CONCEPT(x) ⇔ S(A,B, …)
  where S(A,B,…) is a sentence built using the observable predicates
- The set of all hypotheses is called the hypothesis space H
- An hypothesis h agrees with an example if it gives the correct value of CONCEPT

Inductive Learning Scheme



Size of Hypothesis Space

- n observable predicates
- 2^n entries in truth table defining CONCEPT and each entry can be filled with True or False
- In the absence of any restriction (bias), there are 2^{2^n} hypotheses to choose from
- n = 6 → 2×10^{19} hypotheses!

Multiple Inductive Hypotheses

- Deck of cards, with each card designated by [r,s], its rank and suit, and some cards "rewarded"
- Background knowledge KB:
  ((r=1) v … v (r=10)) ⇔ NUM(r)
  ((r=J) v (r=Q) v (r=K)) ⇔ FACE(r)
  ((s=S) v (s=C)) ⇔ BLACK(s)
  ((s=D) v (s=H)) ⇔ RED(s)
- Training set D:
  REWARD([4,C]) ∧ REWARD([7,C]) ∧ REWARD([2,S]) ∧ ¬REWARD([5,H]) ∧ ¬REWARD([J,S])

h_1 ≡ NUM(r) ∧ BLACK(s) ⇔ REWARD([r,s])
h_2 ≡ BLACK(s) ∧ ¬(r=J) ⇔ REWARD([r,s])
h_3 ≡ ([r,s]=[4,C]) ∨ ([r,s]=[7,C]) ∨ [r,s]=[2,S]) ⇔ REWARD([r,s])
h_4 ≡ ¬([r,s]=[5,H]) ∨ ¬([r,s]=[J,S]) ⇔ REWARD([r,s])
agree with all the examples in the training set

Multiple Inductive Hypotheses

- Deck of cards, with each card designated by [r,s], its rank and suit, and some cards "rewarded"

Need for a system of preferences – called a bias – to compare possible hypotheses

$((s=D) \vee (s=H)) \Leftrightarrow RED(s)$

- Training set D:
  $REWARD([4,C]) \wedge REWARD([7,C]) \wedge REWARD([2,S]) \wedge \neg REWARD([5,H]) \wedge \neg REWARD([J,S])$

$h_1 \equiv NUM(r) \wedge BLACK(s) \Leftrightarrow REWARD([r,s])$

$h_2 \equiv BLACK(s) \wedge \neg(r=J) \Leftrightarrow REWARD([r,s])$

$h_3 \equiv ([r,s]=[4,C]) \vee ([r,s]=[7,C]) \vee [r,s]=[2,S]) \Leftrightarrow REWARD([r,s])$

$h_4 \equiv \neg([r,s]=[5,H]) \wedge \neg([r,s]=[J,S]) \Leftrightarrow REWARD([r,s])$

agree with all the examples in the training set

Inductive learning

- Simplest form: learn a function from examples
- *f* is the target function

An example is a pair (*x*, *f(x)*)

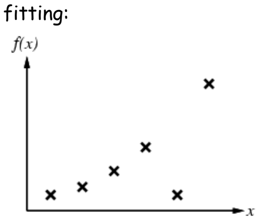
Problem: find a hypothesis *h*
  such that $h \approx f$
  given a training set of examples

(This is a highly simplified model of real learning:
- Ignores prior knowledge
- Assumes examples are given)

Inductive learning method

Construct/adjust *h* to agree with *f* on training set
(*h* is consistent if it agrees with *f* on all examples)

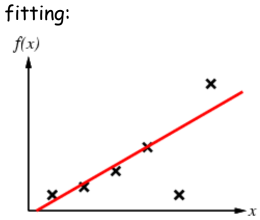
E.g., curve fitting:



Inductive learning method

Construct/adjust *h* to agree with *f* on training set
(*h* is consistent if it agrees with *f* on all examples)

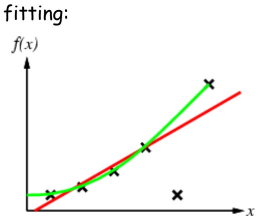
E.g., curve fitting:



Inductive learning method

Construct/adjust *h* to agree with *f* on training set
(*h* is consistent if it agrees with *f* on all examples)

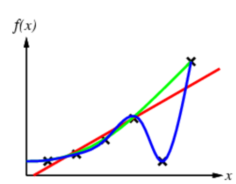
E.g., curve fitting:



Inductive learning method

Construct/adjust *h* to agree with *f* on training set
(*h* is consistent if it agrees with *f* on all examples)

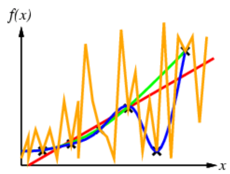
E.g., curve fitting:



Inductive learning method

Construct/adjust *h* to agree with *f* on training set
(*h* is consistent if it agrees with *f* on all examples)

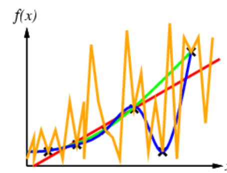
E.g., curve fitting:



Inductive learning method

Construct/adjust *h* to agree with *f* on training set
(*h* is consistent if it agrees with *f* on all examples)

E.g., curve fitting:



Ockham's razor: prefer the simplest hypothesis consistent with data

Notion of Capacity

- It refers to the ability of a machine to learn any training set without error
- A machine with too much capacity is like a botanist with photographic memory who, when presented with a new tree, concludes that it is not a tree because it has a different number of leaves from anything he has seen before
- A machine with too little capacity is like the botanist's lazy brother, who declares that if it's green, it's a tree
- Good generalization can only be achieved when the right balance is struck between the accuracy attained on the training set and the capacity of the machine

→ Keep-It-Simple (KIS) Bias

- **Examples**
  - Use many fewer observable predicates than the training set
  - Constrain the learnt predicate, e.g., to use only "high-level" observable predicates such as NUM, FACE, BLACK, and RED and/or to have simple syntax

Einstein: "A theory must be as simple as possible, but not simpler than this"

- **M**
  - If a hypothesis is too complex it is not worth learning it (data caching does the job as well)
  - There are many fewer simple hypotheses than complex ones, hence the hypothesis space is smaller

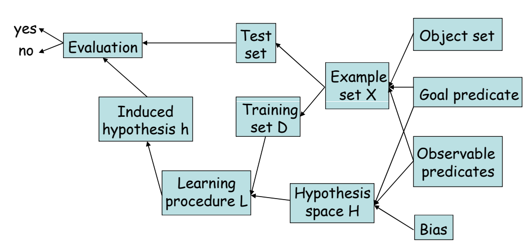
→ Keep-It-Simple (KIS) Bias

- **Examples**

If the bias allows only sentences S that are conjunctions of k << n predicates picked from the n observable predicates, then the size of H is $O(n^k)$

- **Motivation**
  - If a hypothesis is too complex it is not worth learning it (data caching does the job as well)
  - There are many fewer simple hypotheses than complex ones, hence the hypothesis space is smaller

Putting Things Together

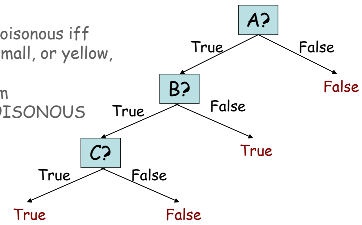


Decision Tree Method

Predicate as a Decision Tree

The predicate CONCEPT(x) ⇔ A(x) ∧ (¬B(x) v C(x)) can be represented by the following decision tree:

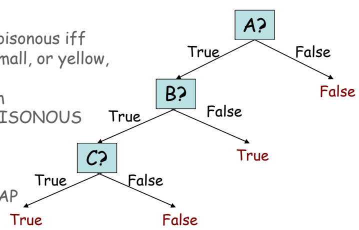
Example:
A mushroom is poisonous iff it is yellow and small, or yellow, big and spotted
- x is a mushroom
- CONCEPT = POISONOUS
- A = YELLOW
- B = BIG
- C = SPOTTED



Predicate as a Decision Tree

The predicate CONCEPT(x) ⇔ A(x) ∧ (¬B(x) v C(x)) can be represented by the following decision tree:

Example:
A mushroom is poisonous iff it is yellow and small, or yellow, big and spotted
- x is a mushroom
- CONCEPT = POISONOUS
- A = YELLOW
- B = BIG
- C = SPOTTED
- D = FUNNEL-CAP
- E = BULKY

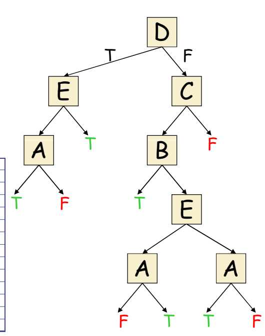


Training Set

Ex. #	A	B	C	D	E	CONCEPT
1	False	False	True	False	True	False
2	False	True	False	False	False	False
3	False	True	True	True	True	False
4	False	False	True	False	False	False
5	False	False	False	True	True	False
6	True	False	True	False	False	True
7	True	False	False	True	False	True
8	True	False	True	False	True	True
9	True	True	True	False	True	True
10	True	True	True	True	True	True
11	True	True	False	False	False	False
12	True	True	False	False	True	False
13	True	False	True	True	True	True

Possible Decision Tree

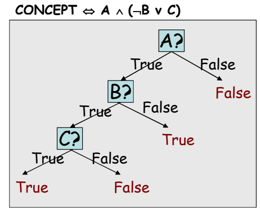
Ex. #	A	B	C	D	E	CONCEPT
1	False	False	True	False	True	False
2	False	True	False	False	False	False
3	False	True	True	True	True	False
4	False	False	True	False	False	False
5	False	False	False	True	True	False
6	True	False	True	False	False	True
7	True	False	False	True	False	True
8	True	False	True	False	True	True
9	True	True	True	False	True	True
10	True	True	True	True	True	True
11	True	True	False	False	False	False
12	True	True	False	False	True	False
13	True	False	True	True	True	True

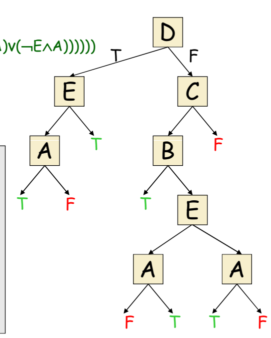


Possible Decision Tree

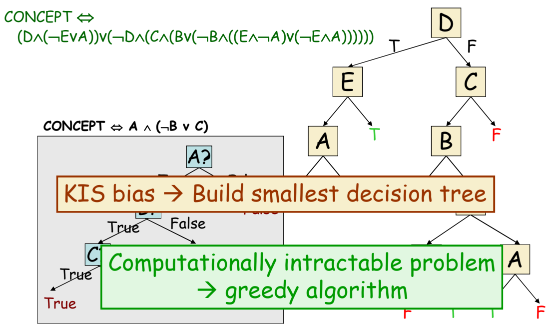
CONCEPT ⇔
(D∧(¬E∨A))∨(¬D∧(C∧(B∨(¬B∧((E∧¬A)∨(¬E∧A))))))

CONCEPT ⇔ A ∧ (¬B v C)





Possible Decision Tree



Picking Best Attribute

- Several different methods
  - Reducing classification error
    - Not covered in the tex
  - Using Information Gain

Getting Started:
Top-Down Induction of Decision Tree

The distribution of training set is:

True: 6, 7, 8, 9, 10,13
False: 1, 2, 3, 4, 5, 11, 12

Ex. #	A	B	C	D	E	CONCEPT
1	False	False	True	False	True	False
2	False	True	False	False	False	False
3	False	True	True	True	True	False
4	False	False	True	False	False	False
5	False	False	False	True	True	False
6	True	False	True	False	False	True
7	True	False	False	True	False	True
8	True	False	True	False	True	True
9	True	True	True	False	True	True
10	True	True	True	True	True	True
11	True	True	False	False	False	False
12	True	True	False	False	True	False
13	True	False	True	True	True	True

Getting Started:
Top-Down Induction of Decision Tree

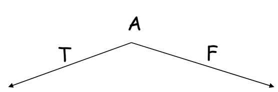
The distribution of training set is:

True: 6, 7, 8, 9, 10,13
False: 1, 2, 3, 4, 5, 11, 12

Without testing any observable predicate, we could report that CONCEPT is False (**majority rule**) with an estimated probability of error P(E) = 6/13

Assuming that we will only include one observable predicate in the decision tree, **which predicate should we test to minimize the probability of error (i.e., the # of misclassified examples in the training set)?** → Greedy algorithm

Assume It's A



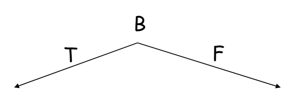
True: 6, 7, 8, 9, 10, 13
False: 11, 12

True: (none)
False: 1, 2, 3, 4, 5

If we test only A, we will report that CONCEPT is True if A is True (majority rule) and False otherwise

→ The number of misclassified examples from the training set is 2

Assume It's B



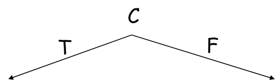
True: 9, 10
False: 2, 3, 11, 12

True: 6, 7, 8, 13
False: 1, 4, 5

If we test only B, we will report that CONCEPT is False if B is True and True otherwise

→ The number of misclassified examples from the training set is 5

Assume It's C

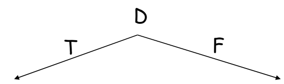


True: 6, 8, 9, 10, 13 | 7
False: 1, 3, 4 | 1, 5, 11, 12

If we test only C, we will report that CONCEPT is True if C is True and False otherwise

→ The number of misclassified examples from the training set is 4

Assume It's D

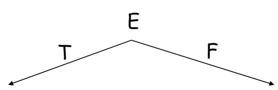


True: 7, 10, 13 | 6, 8, 9
False: 3, 5 | 1, 2, 4, 11, 12

If we test only D, we will report that CONCEPT is True if D is True and False otherwise

→ The number of misclassified examples from the training set is 5

Assume It's E

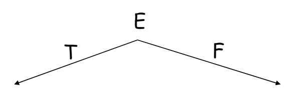


True: 8, 9, 10, 13 | 6, 7
False: 1, 3, 5, 12 | 2, 4, 11

If we test only E we will report that CONCEPT is False, independent of the outcome

→ The number of misclassified examples from the training set is 6

Assume It's E



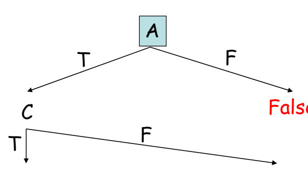
True: 8, 9, 10, 13 | 6, 7
False: 1, 3, 5, 12 | 2, 4, 11

So, the best predicate to test is A

e,
independent of the outcome

→ The number of misclassified examples from the training set is 6

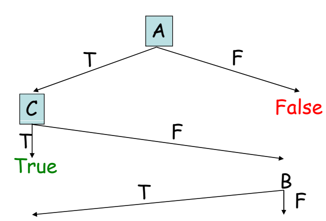
Choice of Second Predicate



True: 6, 8, 9, 10, 13 | 7
False: | 11, 12

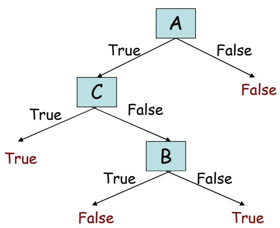
→ The number of misclassified examples from the training set is 1

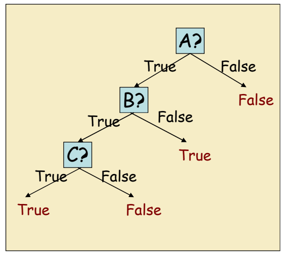
Choice of Third Predicate



True: | 7
False: 11,12

Final Tree

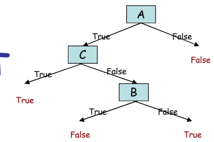




CONCEPT $\Leftrightarrow A \wedge (C \vee \neg B)$

CONCEPT $\Leftrightarrow A \wedge (\neg B \vee C)$

Top-Down Induction of a DT

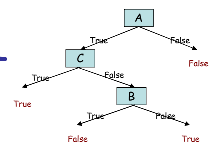


DTL(Δ, Predicates)

1. If all examples in Δ are positive then return True
2. If all examples in Δ are negative then return False
3. If Predicates is empty then return *failure*
4. A ← error-minimizing predicate in Predicates
5. Return the tree whose:
   - root is A,
   - left branch is DTL(Δ^{+A},Predicates-A),
   - right branch is DTL(Δ^{-A},Predicates-A)

Subset of examples that satisfy A

Top-Down Induction of a DT



DTL(Δ, Predicates)

1. If all examples in Δ are positive then return True
2. If all examples in Δ are negative then return False
3. If Predicates is empty then return *failure*
4. A ← error-minimizing predicate in Predicates
5. Return the tree whose:
   - root is A,
   - left branch is DTL(Δ^{+A},Predicates-A),
   - right branch is DTL(Δ^{-A},Predicates-A)

Noise in training set!
May return majority rule,
instead of failure

Comments

- Widely used algorithm
- Greedy
- Robust to noise (incorrect examples)
- Not incremental

Using Information Theory

- Rather than minimizing the probability of error, many existing learning procedures minimize the expected number of questions needed to decide if an object x satisfies CONCEPT
- This minimization is based on a measure of the "quantity of information" contained in the truth value of an observable predicate
- See R&N p. 659-660

Learning decision trees

Problem: decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Attribute-based representations

- Examples described by attribute values (Boolean, discrete, continuous)
- E.g., situations where I will/won't wait for a table:

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

- Classification of examples is positive (T) or negative (F)
-

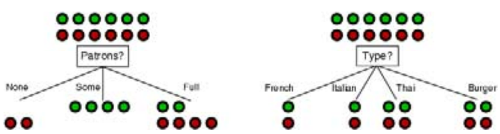
Decision tree learning

- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree

```
function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MODE(examples)
  else
     best ← CHOOSE-ATTRIBUTE(attributes, examples)
     tree ← a new decision tree with root test best
     for each value v_i of best do
        examples_i ← {elements of examples with best = v_i}
        subtree ← DTL(examples_i, attributes − best, MODE(examples))
        add a branch to tree with label v_i and subtree subtree
     return tree
```

Choosing an attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



- *Patrons?* is a better choice

Using information theory

- To implement `Choose-Attribute` in the DTL algorithm
- Information Content (Entropy):

  $I(P(v_1), \ldots, P(v_n)) = \Sigma_{i=1} -P(v_i) \log_2 P(v_i)$
- For a training set containing *p* positive examples and *n* negative examples:

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

Information gain

- A chosen attribute *A* divides the training set *E* into subsets $E_1, \ldots, E_v$ according to their values for *A*, where *A* has v distinct values.

$$remainder(A) = \sum_{i=1}^{v} \frac{p_i+n_i}{p+n} I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$$

- Information Gain (IG) or reduction in entropy from the attribute test:

$$IG(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - remainder(A)$$

- Choose the attribute with the largest IG

Information gain

For the training set, $p = n = 6$, *I(6/12, 6/12) = 1* bit

Consider the attributes *Patrons* and *Type* (and others too):

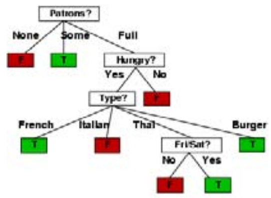
$$IG(Patrons) = 1 - [\frac{2}{12}I(0,1) + \frac{4}{12}I(1,0) + \frac{6}{12}I(\frac{2}{6},\frac{4}{6})] = .0541 \text{ bits}$$

$$IG(Type) = 1 - [\frac{2}{12}I(\frac{1}{2},\frac{1}{2}) + \frac{2}{12}I(\frac{1}{2},\frac{1}{2}) + \frac{4}{12}I(\frac{2}{4},\frac{2}{4}) + \frac{4}{12}I(\frac{2}{4},\frac{2}{4})] = 0 \text{ bits}$$

Patrons has the highest IG of all attributes and so is chosen by the DTL algorithm as the root

Example contd.

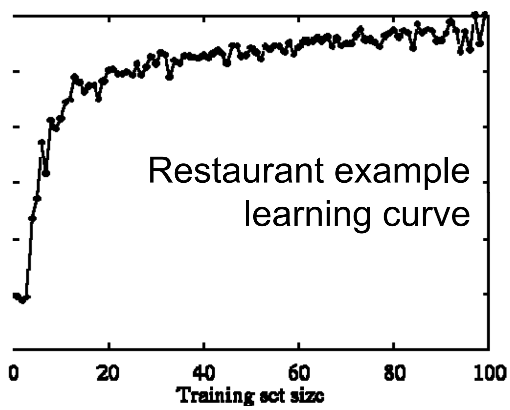
- Decision tree learned from the 12 examples:



- Substantially simpler than “true” tree---a more complex hypothesis isn’t justified by small amount of data

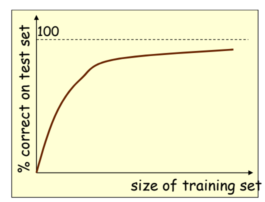
Evaluation methodology

- Standard methodology:
  - 1. Collect a large set of examples (all with correct classifications)
  - 2. Randomly divide collection into two disjoint sets: training and test
  - 3. Apply learning algorithm to training set giving hypothesis H
  - 4. Measure performance of H w.r.t. test set
- Important: keep the training and test sets disjoint!
- To study the efficiency and robustness of an algorithm, repeat steps 2-4 for different training sets and sizes of training sets
- If you improve your algorithm, start again with step 1 to avoid evolving the algorithm to work well on just this collection



Miscellaneous Issues

- Assessing performance:
  - Training set and test set
  - Learning curve

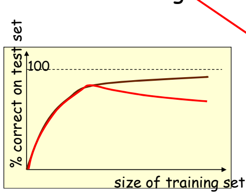


Typical learning curve

Miscellaneous Issues

- Assessing performance:
  - Training set and test set
  - Learning curve
- Overfitting

Risk of using irrelevant observable predicates to generate a hypothesis that agrees with all examples in the training set



Miscellaneous Issues

- Assessing performance:
  - Training set and test set
  - Learning curve
- Overfitting
  - Tree pruning

Risk of using irrelevant observable predicates to generate an hypothesis that agrees with all examples in the training set

Terminate recursion when # errors / information gain is small

Miscellaneous Issues

- Assessing performance:
  - Training set and test set
  - Learning curve
- Overfitting
  - Tree pruning

Risk of using irrelevant observable predicates to ...is ...mples

The resulting decision tree + majority rule may not classify correctly all examples in the training set

Terminate recursion when # errors / information gain is small

Miscellaneous Issues

- Assessing performance:
  - Training set and test set
  - Learning curve
- Overfitting
  - Tree pruning
- Incorrect examples
- Missing data
- Multi-valued and continuous attributes

Extensions of the decision tree learning algorithm

- Using gain ratios (not covered in the text)
- Real-valued data
- Noisy data and overfitting
- Generation of rules
- Setting parameters
- Cross-validation for experimental validation of performance
- C4.5 is an extension of ID3 that accounts for unavailable values, continuous attribute value ranges, pruning of decision trees, rule derivation, and so on

Using gain ratios

- The information gain criterion favors attributes that have a large number of values
  - If we have an attribute D that has a distinct value for each record, then Info(D,T) is 0, thus Gain(D,T) is maximal
- To compensate for this Quinlan suggests using the following ratio instead of Gain:

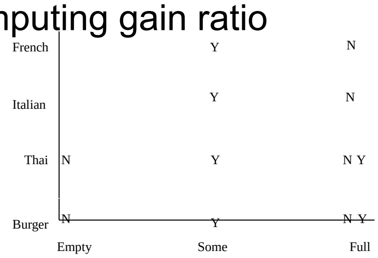
  GainRatio(D,T) = Gain(D,T) / SplitInfo(D,T)
- SplitInfo(D,T) is the information due to the split of T on the basis of value of categorical attribute D

  SplitInfo(D,T) = I(|T1|/|T|, |T2|/|T|, .., |Tm|/|T|)

where {T1, T2, .. Tm} is the partition of T induced by value of D

Computing gain ratio

- I(T) = 1
- I (Pat, T) = .47
- I (Type, T) = 1



Gain (Pat, T) =.53
Gain (Type, T) = 0

SplitInfo (Pat, T) = - (1/6 log 1/6 + 1/3 log 1/3 + 1/2 log 1/2) = 1/6*2.6 + 1/3*1.6 + 1/2*1 = 1.47

SplitInfo (Type, T) = 1/6 log 1/6 + 1/6 log 1/6 + 1/3 log 1/3 + 1/3 log 1/3 = 1/6*2.6 + 1/6*2.6 + 1/3*1.6 + 1/3*1.6 = 1.93

GainRatio (Pat, T) = Gain (Pat, T) / SplitInfo(Pat, T) = .53 / 1.47 = .36

GainRatio (Type, T) = Gain (Type, T) / SplitInfo (Type, T) = 0 / 1.93 = 0

Real-valued data

- Select a set of thresholds defining intervals
- Each interval becomes a discrete value of the attribute
- Use some simple heuristics…
  - always divide into quartiles
- Use domain knowledge…
  - divide age into infant (0-2), toddler (3 - 5), school-aged (5-8)
- Or treat this as another learning problem
  - Try a range of ways to discretize the continuous variable and see which yield “better results” w.r.t. some metric
  - E.g., try midpoint between every pair of values

Noisy data and overfitting

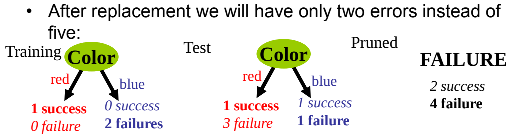
- Many kinds of “noise” can occur in the examples:
  - Two examples have same attribute/value pairs, but different classifications
  - Some values of attributes are incorrect because of errors in the data acquisition process or the preprocessing phase
  - The classification is wrong (e.g., + instead of -) because of some error
  - Some attributes are irrelevant to the decision-making process, e.g., color of a die is irrelevant to its outcome

Noisy data and overfitting (cont)

- The last problem, irrelevant attributes, can result in overfitting the training example data.
  - If the hypothesis space has many dimensions because of a large number of attributes, we may find **meaningless regularity** in the data that is irrelevant to the true, important, distinguishing features
  - Fix by pruning lower nodes in the decision tree
  - For example, if Gain of the best attribute at a node is below a threshold, stop and make this node a leaf rather than generating children nodes

Pruning decision trees

- Pruning of the decision tree is done by replacing a whole subtree by a leaf node
- The replacement takes place if a decision rule establishes that the expected error rate in the subtree is greater than in the single leaf. E.g.,
  - Training: one training red success and two training blue failures
  - Test: three red failures and one blue success
  - Consider replacing this subtree by a single Failure node.
- After replacement we will have only two errors instead of five:



Cross-Validation to Reduce Overfitting

- Estimate how well each hypothesis will predict unseen data.
- Set aside some fraction of the known data, and use it to test the prediction performance of a hypothesis induced from the remaining data.
- K-fold cross-validation means that you run k experiments, each time setting aside a different 1/k of the data to test on, and average the results.
- Use to decide if pruning method is appropriate.
- Need to test again on really unseen data.

Converting decision trees to rules

- It is easy to derive a rule set from a decision tree: write a rule for each path in the decision tree from the root to a leaf
- In that rule the left-hand side is easily built from the label of the nodes and the labels of the arcs
- The resulting rules set can be simplified:
  - Let LHS be the left hand side of a rule
  - Let LHS' be obtained from LHS by eliminating some conditions
  - We can certainly replace LHS by LHS' in this rule if the subsets of the training set that satisfy respectively LHS and LHS' are equal
  - A rule may be eliminated by using metaconditions such as “if no other rule applies”

Applications of Decision Tree

- Medical diagnostic / Drug design
- Evaluation of geological systems for assessing gas and oil basins
- Early detection of problems (e.g., jamming) during oil drilling operations
- Automatic generation of rules in expert systems

How well does it work?

- Many case studies have shown that decision trees are at least as accurate as human experts.
  - A study for diagnosing breast cancer had humans correctly classifying the examples 65% of the time; the decision tree classified 72% correct
  - British Petroleum designed a decision tree for gas-oil separation for offshore oil platforms that replaced an earlier rule-based expert system
  - Cessna designed an airplane flight controller using 90,000 examples and 20 attributes per example

Summary: Decision tree learning

- Inducing decision trees is one of the most widely used learning methods in practice
- Can out-perform human experts in many problems
- Strengths include
  - Fast
  - Simple to implement
  - Can convert result to a set of easily interpretable rules
  - Empirically valid in many commercial products
  - Handles noisy data
- Weaknesses include:
  - Univariate splits/partitioning using only one attribute at a time so limits types of possible trees
  - Large decision trees may be hard to understand
  - Requires fixed-length feature vectors
  - Non-incremental (i.e., batch method)