Learning

Definition

"taking advantage of experience to improve performance"

do more tasks
do tasks more efficiently

"adaptation to environment"

"adjustment of plans"

"acquisition of concepts"
Learning

Definition

Basic Cognitive Architecture

receptors

transducers  interpreter

effectors  long-term memory

interpreter

active memory

process cycle
Learning

Long Term Memory

maintains sets of beliefs
provides means for access

Learning

Operational Definition

adaptive change in long-term memory
based upon history of states of
active and long-term memory
symbolic learning

adaptive change in transducers
based upon input regularities
subsymbolic learning
Learning

What can we learn?

how to better represent the environment

new symbols

new associations between symbols

classification of instances of situations as corresponding to symbols

how to make better action choices

new condition-action rules (based on new concepts)

reinforcement learning action preferences tied to states
Learning

Concept Acquisition

What is a class?

a subset of object instances
(extensional form)

What is a concept?

a description of a class of objects
(intensional form)
which can be used to classify new instances

Motivation

Why concept learning?

input -> concept -> symbol -> knowledge

program (behavior) invariance
access to relevant knowledge
better action choices
Issue

representation of instances of concepts

Learning

instance language

feature vectors
attribute-value pairs

sets of propositions

structured representations

concept language

constraint language over instance language

probabilistic dependencies

output language

concept => symbol for the class (probability)
Concept Representation

symbolic

instances

decision/discrimination trees

rules
  list, set

statistical

support vector machine

neural networks

probabilistic

probabilistic models
  (naive Bayes, Bayesian networks)
Instance Learning

concept represented as a
set of representative instances

classification based on k-nearest neighbors

Issues

how to place instances in a space of
some fixed dimensionality

how to measure distances

how many instances to remember

which instances to remember

non-contiguous or non-separable concepts

how to locate nearest neighbors
Learning

Concept Induction

Complexity

Suppose have n dimensions, each with k possible values.

How many possible instances?

\[ k^n \]

How many possible classes?
(possible subsets of instances)

\[ k^n \]

\[ 2 \]

From a set or sequence of instances, must determine a concept describing one of the possible classes.
Learning

Concept Induction

How can we deal with the inherent complexity of the problem?

Introduce Representation Bias

Instance and Concept Language Restrictions

Introduce Search Heuristics

rules
  Ordered Concept Space
  search cut-offs

decision trees
  which feature to select

weighted network
  Neural Networks
  how to update arc weights

Avoid Search

Bayesian model of data

Instances
Learning

Biased Concept Languages

a priori restriction on search space

suppose 2 dimensions, x and y,
each with eight values

conjunctions
conjunction of attribute-values
(x . 3) and (y . 4)

implicit disjunction
conjunction of attribute-value ranges
(x . [3-5]) and (y . [4-6])

internal disjunction
conjunction of attribute-value disjuncts
(x . [3v7]) and (y . [2v4v6])

limited disjunction
up to k disjuncts (rules)

(((x . 3) and (y . 4)) or (x . 5))
Learning

Bias

How significant are these restrictions on concept languages in reducing the complexity of induction?

Bias Strength

inductive gain

reduction in number of bits needed to represent set of possibilities

\[ \log_2 |\text{total set}| - \log_2 |\text{biased set}| \]

(classes) (concepts)
Learning

inductive gain

Example

2 features, 8 values each

conjunctive concepts

80 possible concepts (64 + 8 + 8)

64 - 7 == 57 bits

implicit disjunction

(sum 8)^2 == 1296 concepts = 11 bits

64 - 11 == 53 bits

internal disjunction

2^8 x 2^8 = 64k concepts = 16 bits

64 - 16 == 48 bits
Learning

Concept Induction

Concept representation restrictions typically eliminate much of the problem.

The rest must be solved by search... willingness to accept inexact concepts

Are there any ways to guide the search?

Search Space Structure

Decision Trees

Rules --- Candidate Elimination
Discrimination Learning

learn to discriminate an instance
    into one of a set of predefined classes

Decision Trees

internal nodes
    test nodes
    value of an attribute (otherwise)

leaf nodes
    result nodes

Methods

incremental (on-line)
global (off-line)
Incremental Discrimination Learning

update tree as instances encountered
combine using with learning of tree

EPAM

get-instance
until reach leaf
   apply test
   and take result branch
if leaf empty
   then say "don't know"
      fill in leaf with correct answer
else say class name $c$ indicated by leaf
   if correct then no change
   else if on otherwise branch
      add class name
         as leaf for test value
         fitting the instance
   else replace leaf with test
      {select feature
       add class name
       as leaf for test value
       fitting the instance}
   make $c$ the otherwise result
Incremental
Discrimination Learning

Example

features
color, size, shape, texture, owner

first instance (red, large, round, rough, art)
say "don't know", correct is "great"
change leaf node to “great”

second instance (red, small, square, rough, art)
say "great", but correct is "so-so"
replace leaf with test for size
  if small has answer "so-so"
  otherwise “great”

third instance (green, small, square, smooth, art)
say “so-so”, correct is “good”
replace leaf with test for color
  if green has answer "good"
  otherwise “so-so”

fourth instance is (red, large, square, smooth, ted)
say “great” and answer is “great” so no change
Discrimination Learning

EPAM Performance

impact of order of instances
can produced very skewed trees

better if repeated items are not grouped

learn only when an error

approximate model of human discrimination learning
Decision Tree Learning

global method

have all instances

split into
  training set
  testing set

learning algorithm

create root node, with all training set instances;
put root node in queue;
until [empty queue]
take first node;
if [all elements do not have same output class]
  select an attribute
  create a child for each value,
    with associated instances
  put children in queue

How do we select the next attribute?
Decision Tree

attribute selection

situation
at some leaf of the tree, where set of
instances matching conditions on
the path to the leaf are in
more than one class
and
there are unused attributes

select which attribute to use to create test node

try them all and see how they
distribute instances to the new leaves

select best according to some measure

number of empty leaves
number of "complete" leaves
evenness/skew of distribution
information gain
Decision Trees

information gain

information in a set of instances
function of
number of classes
probabilistic distribution of instances

\[ I_{\text{old}} = \sum -p_i \times \log_2 p_i \]
where \( p_i \) is probability of instance being in class \( i \) in set of instances at the leaf

if add an attribute \( a \), create \( k \) new leaves each with some probability \( p_k \)
and with some information \( I_k \)

(expected) information after adding attribute \( a \) is

\[ I_{\text{new}} = \sum p_k I_k \]

information gain is:

\[ I_{\text{old}} - I_{\text{new}} \]

choose attribute with largest information gain
Decision Trees

Issues
overlearning

Response
introduce acceptance threshold

instead of a leaf having instances of
only one class
accept a node as leaf when a percentage
greater than threshold is of one class

maintain a distribution representation at each node

when a query lacks an attribute or
has a new attribute value
select reply according to distribution at a node

pruning
remove parts of tree that do not produce
a significant information gain
Concept Evaluation

simpler concept is better
"Occam’s razor"

better classification performance is better
(on non-training set instances)

instances split into
training and non-training sets

concept is a generalization
beyond what has been seen

overlearning can be a problem
want our experiences to be generalizable

learning can only be as good as examples
are representative of domain
Classification Learning:  
List of Conjunctive Rules

data covering algorithms

let instance-set = all instances  
let L = empty list;  
until [instance-set is empty]  
  find a conjunctive condition C that for a set  
    of purely positive or negative instances I;  
  append C to list L with appropriate response;  
  let instance-set = instance-set - I;

How can we find a set of conditions?

  a condition is [feature == value]  
    (or >=, <=, etc. if numeric)

1. Start with a single condition and  
   add others as necessary

2. Start from an instance and remove or  
   generalize features as possible
Decision Stumps and Ensemble Learning

Decision Stump

decision tree with a single test node (as root) or to fixed, small depth

Ensemble Learning

partition data into k random subsets

find best decision stump for each (or result of any learning algorithm)

use these to vote on decision majority wins

assuming independence of the classifiers, then the chance correct
(1- chance majority is wrong)

5 classifiers each with 0.2 error => majority has about 0.05 error

5 classifiers each with 0.1 error => majority has less than 0.01 error
Boosting

perform learning several times,
weighting instances greater
if have been misclassified

ADA Boost learning algorithm

weight all examples equally

loop k times

learn a classifier (e.g., decision tree or stump)
with its value being percent correct on test set
based on weights

reweight examples.. more weight to error instances

return the k classifiers,
each weighted by its percent correct

to use

make decisions based on weighted majority
Classification Learning

On-line Search for Conjunctive Rule Concept

Ordered Concept Space

\[ \text{concept1} > \text{concept2} \quad \text{(i.e., more general than)} \]

iff

class described by concept1
contains the class
\[
\text{described by concept2}
\]

concepts arranged in a lattice
most general \( === \) all instances (\( U \))
least general \( === \) no instances (\( \emptyset \))
not all concepts comparable

\[
U
\]

\[
\emptyset
\]
Learning

Search

Ordered Concept Space

allows "legal" cut-offs/pruning

allows search to be a process of containment between upper and lower bounds within search space

Candidate Elimination Search

\[ S = \{ c \mid c \text{ is consistent with observations } \{O\}, \text{ and there does not exist } c' \text{ such that } c' < c \text{ and } c' \text{ is consistent with } \{O\} \} \]

\[ G = \{ c \mid c \text{ is consistent with observations } \{O\}, \text{ and there does not exist } c' \text{ such that } c < c' \text{ and } c' \text{ is consistent with } \{O\} \} \]
Learning

Version Space

cconcept $c$ is consistent with observations $\{O\}$

iff

does not exist an $o$ in $\{O\}$ such that $o$ is in $c$
yet is not in class to be described

does not exist an $o$ in $\{O\}$ such that $o$ not in $c$
yet is in class to be described

\[ U \]
\[ G \]
\[ \text{version space} \]
\[ S \]
\[ \emptyset \]
Version Space Search

What operators does it apply when sees positive/negative instance(s)?

**positive instance o**

(generalize o G S)

G = eliminate those that do not match o

S = generalize those not containing o
     a minimum amount to contain o
     keeping "under" G

**negative instance o**

(specialize o G S)

G = specialize those containing o
     a minimum amount to not contain o
     keeping "above" S

S = eliminate those that do match o
Version Space Search

(implicit disjunction bias)

2-dimensional example

Looking for rules of form:

(if ((val x (x1 x2))
     (val y (y1 y2))) then (assert c))

on slides, represented as:

((x1-x2), (y1-y2))

Initialization

G: ((if true then (assert c)))
   always succeeds

S: ((if false then (assert c)))
   never succeeds
Version Space Search

(implicit disjunction bias)

\{O\} = \{1-6, 1-3, 4-6, 1-8\}

S = \{(5-6, 2-3)\}

Suppose get positive instance, (4, 6)

\{O\} = \{(4-6, 1-8)\}

S = \{(4-6, 2-6)\}
Version Space Search
(implicit disjunction bias)

\[ \{O\} = \ldots \ldots + \ldots \]
\[ G = \{(1-6, 1-3), (4-6, 1-8)\} \]
\[ S = \{(5-6, 2-3)\} \]

Suppose get negative instance, (6, 5)
Version Space Search
(implicit disjunction bias)

\[ \text{G} = \{(1-6, 1-3), (4-6, 1-8)\} \]
\[ \text{S} = \{(5-6, 2-3)\} \]

Suppose get positive instance, as shown
\[ (2, 6) \]

\[ \text{G} = \{} \]
\[ \text{S} = \{} \]

The implicit disjunction bias makes it impossible to represent this class
Version-Space Search

Properties

no memory required for training instances

sensitivity to error in observation data
as originally proposed

recognizes bias failures
could introduce expanded bias
previous trials consistent
with S and G when fails
Naive Bayes Learning

Bayes Theorem

given data consisting of attribute vectors of length n

\[ A = [a_1, a_2, ..., a_n] \]

each classified in one of m classes, C1, C2, ..., Cm

\[
P(C_j | [a_1, a_2, ..., a_n]) =
\]
\[
P(C_j) \frac{P([a_1, a_2, ..., a_n] | C_j)}{P([a_1, a_2, ..., a_n])} =
\]
\[
P(C_j) P([a_1, a_2, ..., a_n] | C_j),
\]
as \( P([a_1, a_2, ..., a_n]) \) equal for all classes

Naive Assumption

all attributes are independent for a given class \( C_j \),
therefore

\[
P([a_1, a_2, ..., a_n] | C_j) = \pi P(a_k | C_j), \text{ over all } k
\]
Naive Bayes Classification

\[ P(C_j) \text{ can be estimated from data as percent of instances that are classified } C_j \]

\[ P(a_k \mid C_j) \text{ can be estimated from data as percent of instances of } C_j \text{ having value } a_k \text{ for } k\text{th attribute} \]

Given instance \( l \) as attribute vector

\[ l = [l_{a_1}, l_{a_2}, \ldots, l_{a_n}] \]

choose class that maximizes

\[
P(C_j) \prod_k P(l_{a_k} \mid C_j) \]

over all classes \( C_j \)
Learning

Example

<table>
<thead>
<tr>
<th>RID</th>
<th>age</th>
<th>income</th>
<th>student</th>
<th>credit_rating</th>
<th>Class: buys_computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>youth</td>
<td>high</td>
<td>no</td>
<td>fair</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>youth</td>
<td>high</td>
<td>no</td>
<td>excellent</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>middle_aged</td>
<td>high</td>
<td>no</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>senior</td>
<td>medium</td>
<td>no</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>senior</td>
<td>low</td>
<td>yes</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>6</td>
<td>senior</td>
<td>low</td>
<td>yes</td>
<td>excellent</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>middle_aged</td>
<td>low</td>
<td>yes</td>
<td>excellent</td>
<td>yes</td>
</tr>
<tr>
<td>8</td>
<td>youth</td>
<td>medium</td>
<td>no</td>
<td>fair</td>
<td>no</td>
</tr>
<tr>
<td>9</td>
<td>youth</td>
<td>low</td>
<td>yes</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>senior</td>
<td>medium</td>
<td>yes</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>11</td>
<td>youth</td>
<td>medium</td>
<td>yes</td>
<td>excellent</td>
<td>yes</td>
</tr>
<tr>
<td>12</td>
<td>middle_aged</td>
<td>medium</td>
<td>no</td>
<td>excellent</td>
<td>yes</td>
</tr>
<tr>
<td>13</td>
<td>middle_aged</td>
<td>high</td>
<td>yes</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>14</td>
<td>senior</td>
<td>medium</td>
<td>no</td>
<td>excellent</td>
<td>no</td>
</tr>
</tbody>
</table>
Bayes Learning

Example

Classify:

(\text{age}=\text{youth}, \text{income}=\text{medium}, \text{student}=\text{yes}, \text{credit}=\text{fair})

P(\text{yes}) = \frac{9}{14} = .643; \ P(\text{no}) = 0.357

P(\text{youth|yes}) = 0.222; \ P(\text{youth|no}) = 0.60
P(\text{medium|yes}) = 0.444; \ P(\text{medium|no}) = 0.40
P(\text{yes|yes}) = 0.667; \ P(\text{yes|no}) = 0.20
P(\text{fair|yes}) = 0.667; \ P(\text{fair|no}) = 0.40

P(\text{yes|X}) = .643*.222*.444*.667*.667 = 0.0282

P(\text{no|X}) = .357*.6*.4*.2*.4 = 0.0069

one could normalize over their sum $0.0351$

$P(\text{yes|X}) = 0.80 \quad P(\text{no|X}) = 0.20$
Unsupervised Learning

no class labels or feature given

Conjunctive Association Rules

frequent items and associations

input

set of items --- market basket
feature vector --- an individual

Given a data set D, support(A) is count of occurrences of A, where A is an item set of feature/value set

A== {beer, chips} in a market basket
A == {color/red, size/large} in an individual

support can be expressed as percent of data

Frequent item set: one with support above a minimum support level (a parameter to learning)
Unsupervised Learning

Conjunctive Association Rules

Given an association rule (A => B)
  (where A and B are items sets or feature/value sets),

  confidence(A => B) == support(A and B)/ support(A)

Strong Association Rule:
  An association rule (A => B)
    such that (A and B) is a frequent item set
    and confidence (A => B) is above a predefined minimum confidence level
    (another parameter to learning)

  rules that are correct sufficiently often (75%)
    based on items that occur sufficiently often (2%)
Unsupervised Learning

Conjunctive Association Rules

Task: Given a data set D,
find all Strong Association Rules

Important Fact

If (A and B) is a frequent item set, then
A and B are both frequent item sets.
(a priori or inheritance property)

Algorithm FIS
Start with individual items as item sets
find frequent item sets (FIS$_1$)
Until (FIS$_k$) is empty
find FIS$_{k+1}$ by combining sets from FIS$_k$
and including those that are frequent

Algorithm SAR
Given the frequent item sets and their support
Consider all association rules based on the
frequent item sets and accept those that
have sufficient confidence
Unsupervised Learning

Conjunctive Association Rules (2, .75)

<table>
<thead>
<tr>
<th>TID</th>
<th>List of item IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>1, 2, 15</td>
</tr>
<tr>
<td>T200</td>
<td>2, 14</td>
</tr>
<tr>
<td>T300</td>
<td>2, 13</td>
</tr>
<tr>
<td>T400</td>
<td>1, 2, 14</td>
</tr>
<tr>
<td>T500</td>
<td>1, 13</td>
</tr>
<tr>
<td>T600</td>
<td>2, 13</td>
</tr>
<tr>
<td>T700</td>
<td>1, 13</td>
</tr>
<tr>
<td>T800</td>
<td>1, 2, 13, 15</td>
</tr>
<tr>
<td>T900</td>
<td>1, 2, 13</td>
</tr>
</tbody>
</table>

Scan D for count of each candidate

\[ C_1 \]

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup. count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{11}</td>
<td>6</td>
</tr>
<tr>
<td>{12}</td>
<td>7</td>
</tr>
<tr>
<td>{13}</td>
<td>6</td>
</tr>
<tr>
<td>{14}</td>
<td>2</td>
</tr>
<tr>
<td>{15}</td>
<td>2</td>
</tr>
</tbody>
</table>

Compare candidate support count with minimum support count

\[ L_1 \]

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup. count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{11}</td>
<td>6</td>
</tr>
<tr>
<td>{12}</td>
<td>7</td>
</tr>
<tr>
<td>{13}</td>
<td>6</td>
</tr>
<tr>
<td>{14}</td>
<td>2</td>
</tr>
<tr>
<td>{15}</td>
<td>2</td>
</tr>
</tbody>
</table>

Generate \( C_2 \) candidates from \( L_1 \)

\[ C_2 \]

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup. count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{11, 12}</td>
<td>4</td>
</tr>
<tr>
<td>{11, 13}</td>
<td>4</td>
</tr>
<tr>
<td>{11, 14}</td>
<td>1</td>
</tr>
<tr>
<td>{11, 15}</td>
<td>2</td>
</tr>
<tr>
<td>{12, 13}</td>
<td>4</td>
</tr>
<tr>
<td>{12, 14}</td>
<td>2</td>
</tr>
<tr>
<td>{12, 15}</td>
<td>2</td>
</tr>
<tr>
<td>{13, 14}</td>
<td>0</td>
</tr>
<tr>
<td>{13, 15}</td>
<td>1</td>
</tr>
<tr>
<td>{14, 15}</td>
<td>0</td>
</tr>
</tbody>
</table>

Scan D for count of each candidate

\[ C_3 \]

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup. count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{11, 12, 13}</td>
<td>2</td>
</tr>
<tr>
<td>{11, 12, 15}</td>
<td>2</td>
</tr>
</tbody>
</table>

Compare candidate support count with minimum support count

\[ L_2 \]

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup. count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{11, 12}</td>
<td>4</td>
</tr>
<tr>
<td>{11, 13}</td>
<td>4</td>
</tr>
<tr>
<td>{11, 15}</td>
<td>2</td>
</tr>
<tr>
<td>{12, 13}</td>
<td>4</td>
</tr>
<tr>
<td>{12, 14}</td>
<td>2</td>
</tr>
<tr>
<td>{12, 15}</td>
<td>2</td>
</tr>
</tbody>
</table>

Generate \( C_3 \) candidates from \( L_2 \)

\[ L_3 \]

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup. count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{11, 12, 13}</td>
<td>2</td>
</tr>
<tr>
<td>{11, 12, 15}</td>
<td>2</td>
</tr>
</tbody>
</table>

Unsupervised Learning

Conjunctive Association Rules

What can we find if need support of 2 and confidence of 75%?

At level 2:

I4 => I2
I5 => I2

At level 3:

I1 and I5 => I2
I2 and I5 => I1
Unsupervised Learning

clustering

* * * * * * * *

grouping like data into new classes

issues

distances

symbolic features

scales/normalization

methods

hierarchical

k-means

minimum

model-based

data

noise/outliers
Artificial Neural Networks

basic notions

analogy to nervous system

many interconnected, simple, computing elements
acting in parallel

computing elements == nodes

determine and pass along activation

input activation levels

activation function

output level

links

weight ... transmission coefficient
Neural Nets

computing units

input activation

\[ \text{in}_i = \text{sum of link weighted activations of input connected units} \]

activation function \( a_i \)

step \{0, 1\}

sign \{-1, 1\} around a threshold \( t \)

sigmoid \([0 .. 1]\)

(continuous approximation of step function)

output level

\[ a_i = a_i(\text{in}_i) \]
Perceptrons

Single layer Feed-forward Networks

<table>
<thead>
<tr>
<th>input units</th>
<th>computing/output units</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>o</td>
</tr>
<tr>
<td>0</td>
<td>o</td>
</tr>
<tr>
<td>0</td>
<td>o</td>
</tr>
<tr>
<td>.</td>
<td>o</td>
</tr>
<tr>
<td>.</td>
<td>o</td>
</tr>
<tr>
<td>0</td>
<td>o</td>
</tr>
<tr>
<td>0</td>
<td>o</td>
</tr>
<tr>
<td>0</td>
<td>o</td>
</tr>
<tr>
<td>0</td>
<td>o</td>
</tr>
</tbody>
</table>

Each output can represent a linearly separable function of inputs.

There exists a procedure that can learn any such function.

Gradient decent to reduce error by adjusting weights on links.

Learning procedure converges, if it can, due to strong bias (unimodal metric).

XOR classic, minimal function that can not be realized.
Multilayered Feed-Forward Networks

input units  computing units
  0
  0  o  o
  0  o
  o
  o
  o
  0  o
  0  o  o
  0

nice properties
  efficient learning
  convergence
are lost
  as give up bias..

but arbitrary function can be represented as
  sums of sigmoids
so essentially an intractable problem..
  looking for quick approximations

has gained acceptance as alternative
  to non-linear function regression
Neural Nets

Issues

parameter setting
learning rate
number of "hidden layer" units

affect time and accuracy

recurrent networks
-- feedback nets

Hopfield Nets

Adaptive Resonance Theory

Temporal Delay Learning
Explanation-based Approaches

Knowledge in learning

Goal Concept

Domain Theory (Prior Knowledge)

Instances

====> Operational Concept

Goal Symbol

CUP

holds-liquid       can-be-picked-up       can-drink-from

..........         ...........         ...........

Theory

C11   C12   C21   C22   C31

existing concepts

Instance I1 I2 I3 I4 I5
Explanation-based Learning

Process

establish proof of instance as element
of goal concept through existing concepts

backward chaining
through domain theory

generalize proof by regressing
variables through proof structure

taking lowest-level concepts as primitives

some constants may remain
some variables repeated
Explanation-based Learning

Example

attempt suicide

Goal

try-to-kill(x,x)

Domain Theory

hate(s,t) and possess(s,w) and weapon(w) => try-to-kill(s,t)

depressed(y) => hate(y,y)
recently-divorced(y) => depressed(y)
failed-exam(y) => depressed(y)
......
buy(y,z) => possess(y,z)
borrow(y,z) => possess(y,z)
...handgun(z) => weapon(z)
rifle(z) => weapon(z)
....

Instance

try-to-kill(john,john)  height(john, tall)
recently-divorced(john)  hair-color(john, red)
borrow(john,thing1)  possess(john, corvette)
rifle(thing1)  recently-travelled(john, europe)
Explanation-based Learning

e
t

Example

try-to-kill(john, john)

hate(j, j) possess(j, t1) weapon(t1)

depressed(j) borrow(j, t1) rifle(t1)

recently-divorced(j)

Rule Acquired

recently-divorced(x) and

borrow(x, y) and rifle(y)

==> try-to-kill(x, x)

operationality
design
Explanation-based Learning

domain knowledge provides bias
new concept expressed
  in terms of
  explained by
  existing domain theory

domain knowledge focuses learning
  on features included in domain rules

what is learned???
  all knowledge is already there
  no new domain knowledge
  new rule implicit in prior knowledge

what is role of instance???
  design -vs- concept formation