MALINENI LAKSHMAIAH WOMEN'S ENGINEERING COLLEGE
(Approved by AICTE, Affiliated to JNTUK)(An ISO9001:2008 Certified Institution)

IV B.Tech (Common to CSE and IT), IV-I Semester, R19, Machine Learning Notes, UNIT-III

Prepared by Dr.M.BHEEMALINGAIAH

UNIT-III

Computational Learning Theory: Models of learnability: learning in the limit; probably approximately correct (PAC) learning. Sample complexity for infinite hypothesis spaces, Vapnik-Chervonenkis dimension.
Rule Learning: Propositional and First-Order, Translating decision trees into rules, Heuristic rule induction using separate and conquer and information gain, First-order Horn-clause induction (Inductive Logic Programming) and Foil, Learning recursive rules, Inverse resolution, Golem, and Progol

Content

3.1 Computational Learning Theory

Computational learning theory, or CoLT for a brief, could be a field of study involved with utilizing formal mathematical strategies applied to learning systems. It seeks to utilize the tools of theoretical technology to quantify learning issues. This includes characterizing the problem of learning specific tasks. Computational learning theory is also thought of as an associate extension or relation of applied math or statistical learning theory, or SLT for a brief that uses formal strategies to quantify learning algorithms.

When studying machine learning it is natural to wonder what general laws may govern machine (and nonmachine) learners. Is it possible to identify classes of learning problems that are inherently difficult or easy, independent of the learning algorithm? Can one characterize the number of training examples necessary or sufficient to assure successful learning? How is this number affected if the learner is allowed to pose queries to the trainer, versus observing a random sample of training examples? Can one characterize the number of mistakes that a learner

will make before learning the target function? Can one characterize the inherent computational complexity of classes of learning problems?

Although general answers to all these questions are not yet known, frag- ments of a computational theory of learning have begun to emerge. This chapter presents key results from this theory, providing answers to these questions within particular problem settings. We focus here on the problem of inductively learning an unknown target function, given only training examples of this target func- tion and a space of candidate hypotheses. Within this setting, we will be chiefly concerned with questions such as how many training examples are sufficient to successfully learn the target function, and how many mistakes will the learner make before succeeding. As we shall see, it is possible to set quantitative bounds on these measures, depending on attributes of the learning problem such as:

- the size or complexity of the hypothesis space considered by the learner
- the accuracy to which the target concept must be approximated
- the probability that the learner will output a successful hypothesis
- the manner in which training examples are presented to the learner

For the most part, we will focus not on individual learning algorithms, but rather on broad classes of learning algorithms characterized by the hypothesis spaces they consider, the presentation of training examples, etc. Our goal is to answer questions such as:

- *Sample complexity.* How many training examples are needed for a learner to converge (with high probability) to a successful hypothesis?
- *Computational complexity.* How much computational effort is needed for a learner to converge (with high probability) to a successful hypothesis?
- *Mistake bound.* How many training examples will the learner misclassify before converging to a successful hypothesis?

3.2 Probably Approximately Correct (PAC) Leaner

PAC learning is a framework for the mathematical analysis of machine learning

Goal of PAC: With high probability ("Probably") , the selected hypothesis will have lower error ("Approximately Correct")

Definition: The **true error** (denoted $error_{\mathcal{D}}(h)$) of hypothesis h with respect to target concept c and distribution $\mathcal{D}$ is the probability that h will misclassify an instance drawn at random according to $\mathcal{D}$.

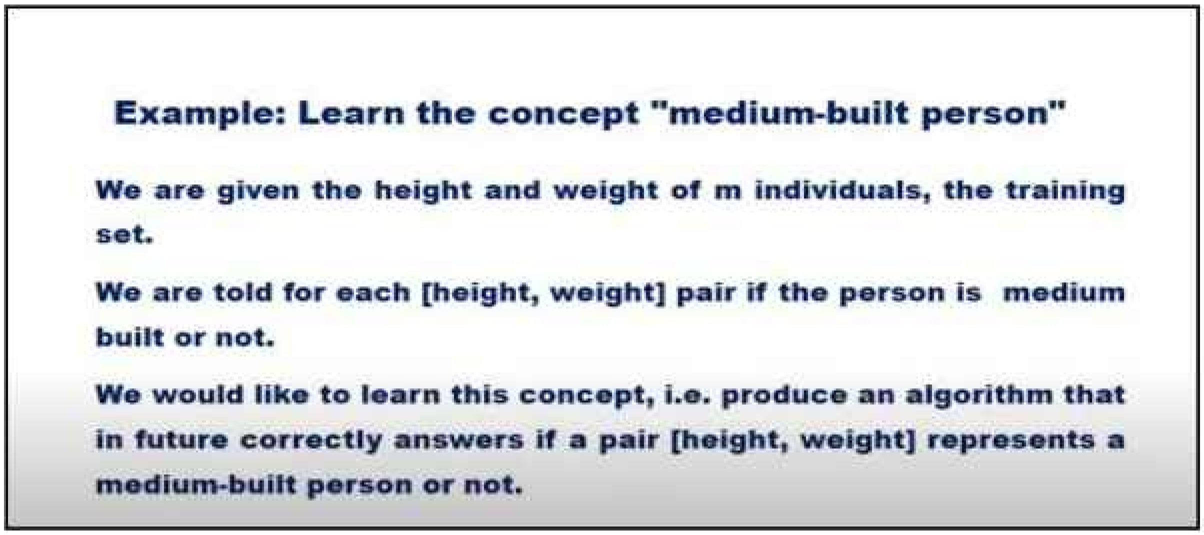
$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[c(x) \neq h(x)]$$

Here the notation $\Pr_{x \in \mathcal{D}}$ indicates that the probability is taken over the instance distribution $\mathcal{D}$.

Definition: Consider a concept class C defined over a set of instances X of length n and a learner L using hypothesis space H. C is **PAC-learnable** by L using H if for all $c \in C$, distributions $\mathcal{D}$ over X, ϵ such that $0 < \epsilon < 1/2$, and δ such that $0 < \delta < 1/2$, learner L will with probability at least $(1 - \delta)$ output a hypothesis $h \in H$ such that $error_{\mathcal{D}}(h) \leq \epsilon$, in time that is polynomial in $1/\epsilon$, $1/\delta$, n, and $size(c)$.

ε and δ parameters

ε gives an upper bound on the error in accuracy with which h approximated (accuracy: 1- ε)

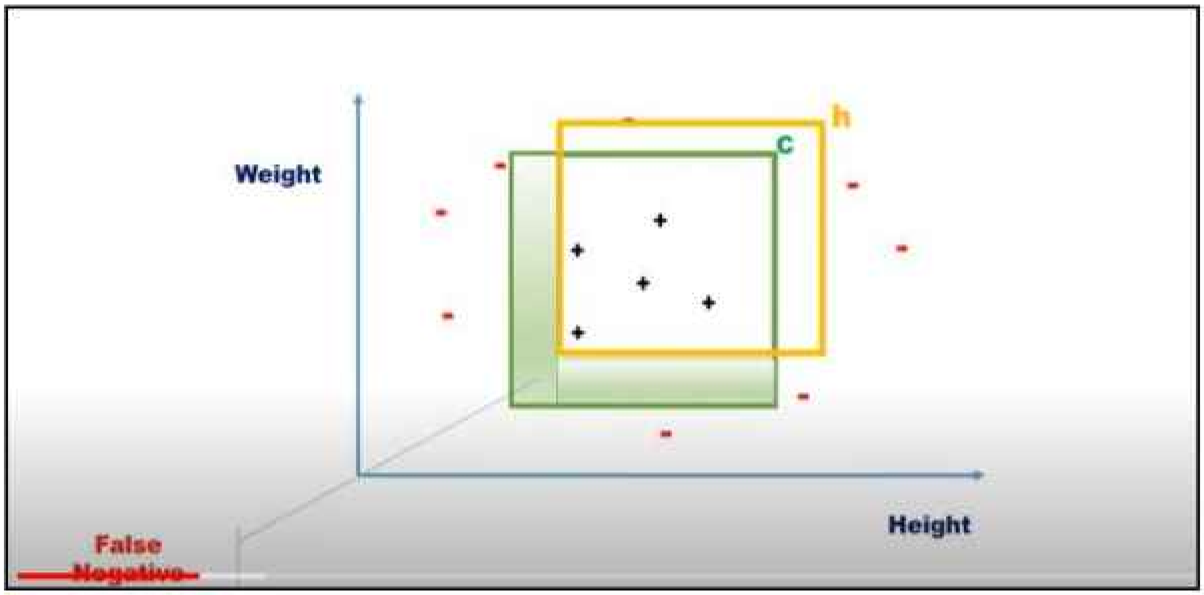


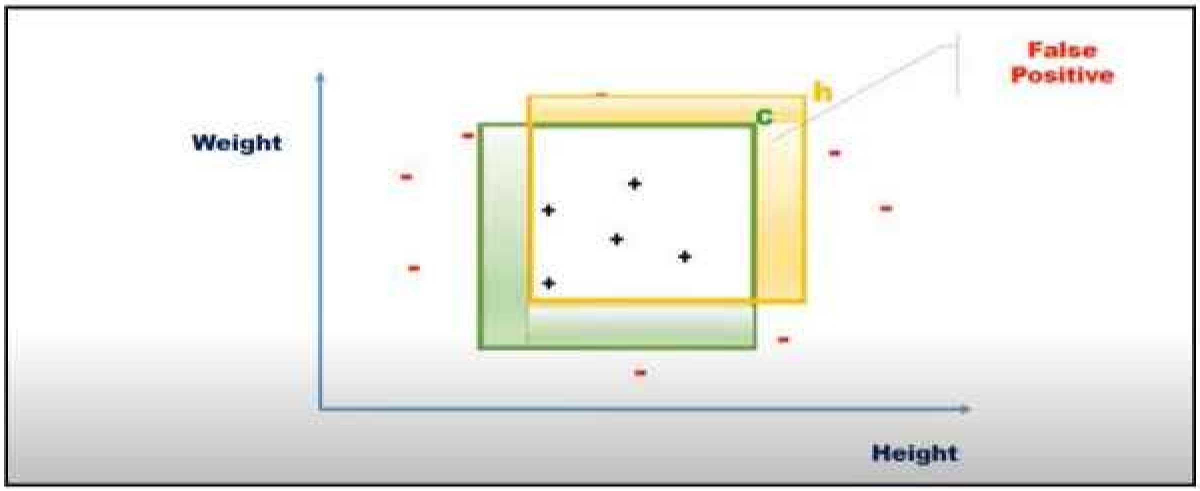
Example: Learn the concept "medium-built person"

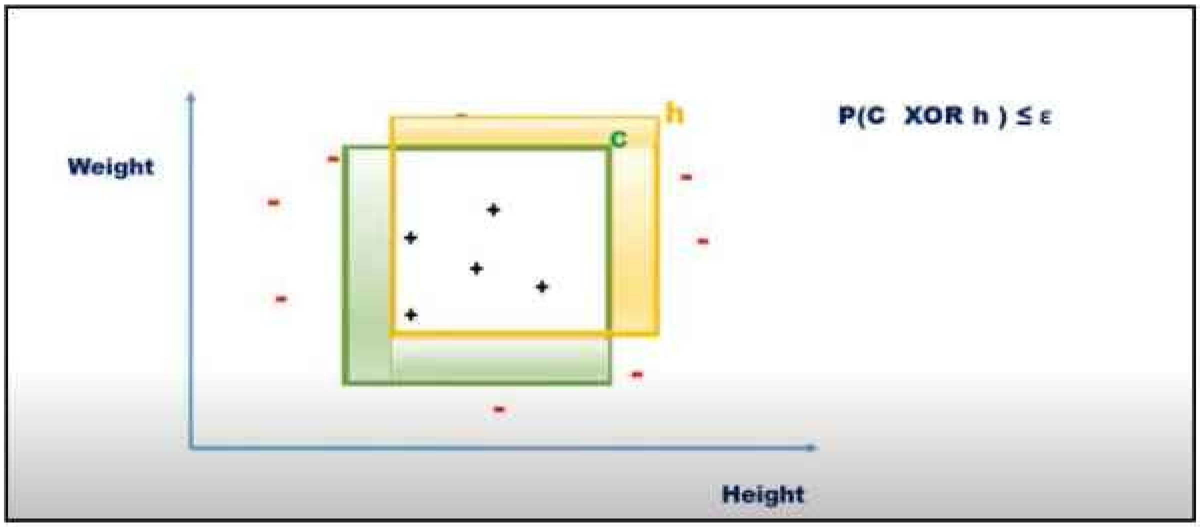
We are given the height and weight of m individuals, the training set.

We are told for each [height, weight] pair if the person is medium built or not.

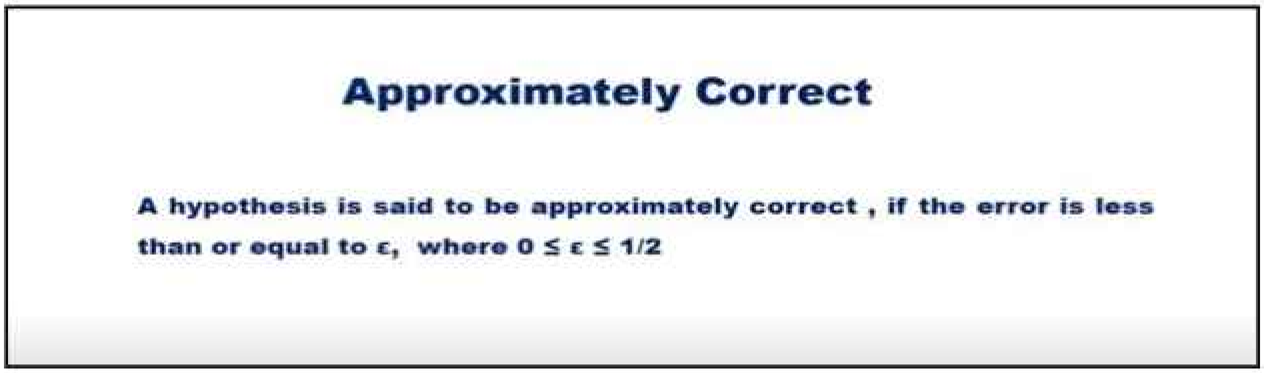
We would like to learn this concept, i.e. produce an algorithm that in future correctly answers if a pair [height, weight] represents a medium-built person or not.





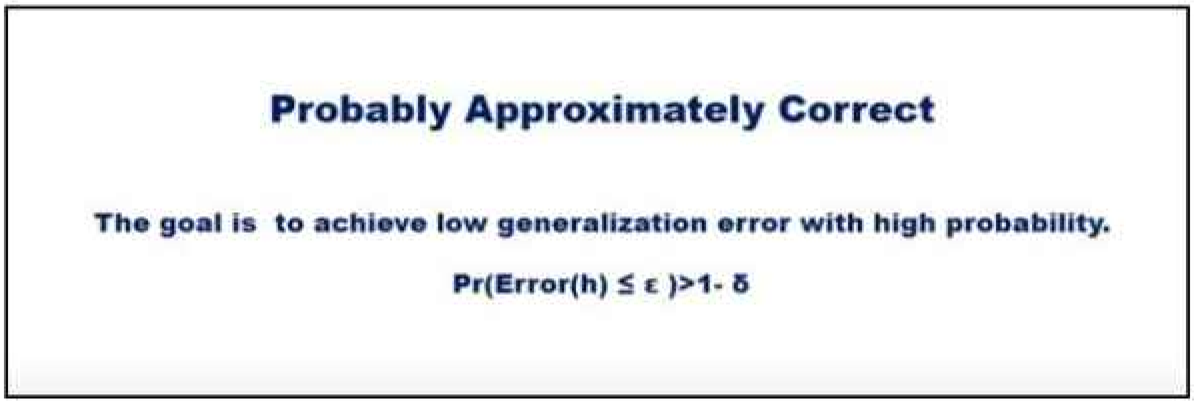


$P(C \text{ XOR } h) \leq \varepsilon$



Approximately Correct

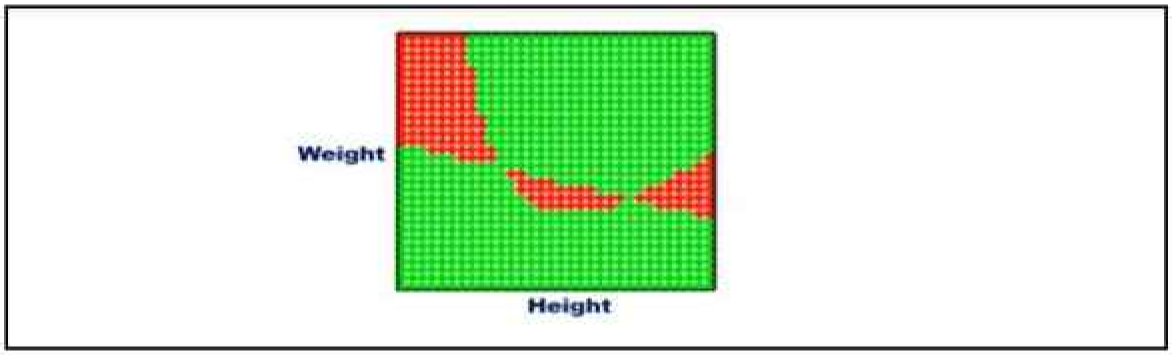
A hypothesis is said to be approximately correct , if the error is less than or equal to ε, where $0 \leq \varepsilon \leq 1/2$

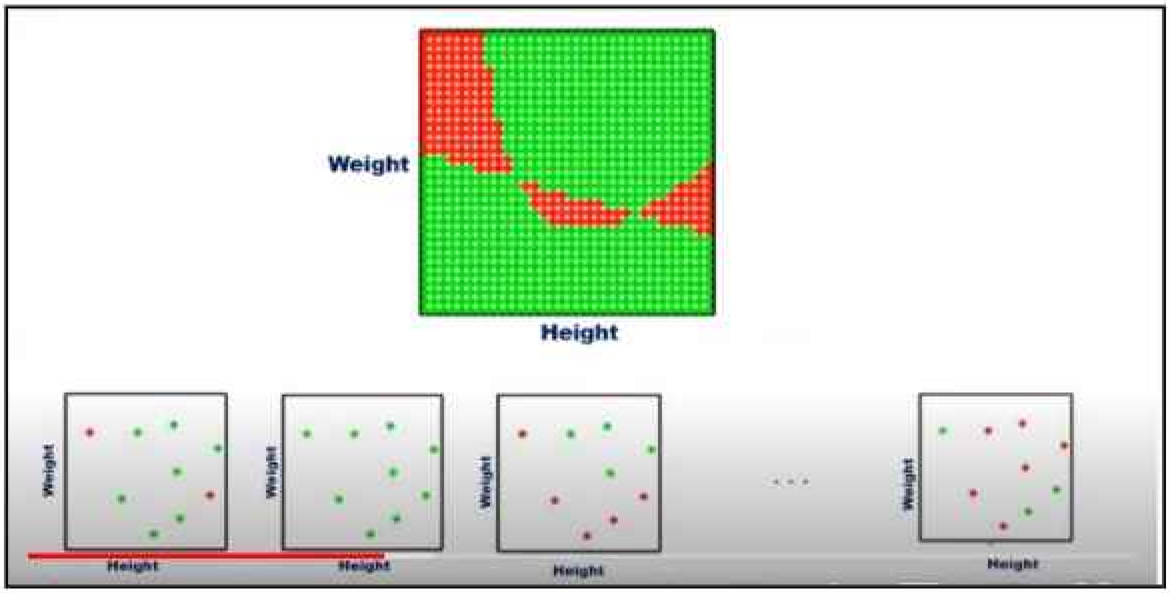


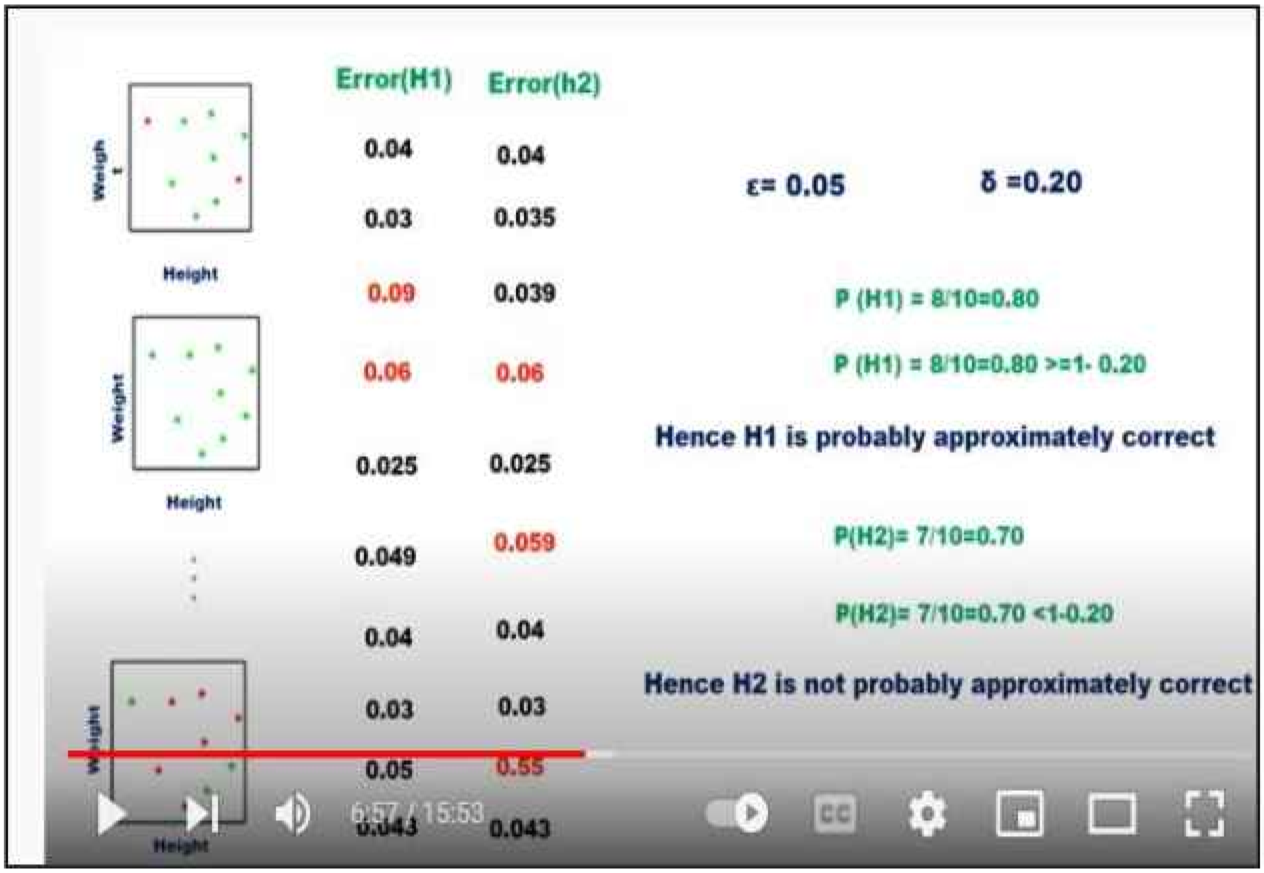
Probably Approximately Correct

The goal is to achieve low generalization error with high probability.

$Pr(Error(h) \leq \varepsilon) > 1 - \delta$


Weight
Height


Weight
Height
Weight
Height
Weight
Height
Weight
Height
. . .
Weight
Height



Error(H1)	Error(h2)
0.04	0.04
0.03	0.035
0.09	0.039
0.06	0.06
0.025	0.025
0.049	0.059
0.04	0.04
0.03	0.03
0.05	0.55
0.043	0.043

$\varepsilon = 0.05$ $\delta = 0.20$

P (H1) = 8/10=0.80

P (H1) = 8/10=0.80 >=1- 0.20

Hence H1 is probably approximately correct

P(H2)= 7/10=0.70

P(H2)= 7/10=0.70 <1-0.20

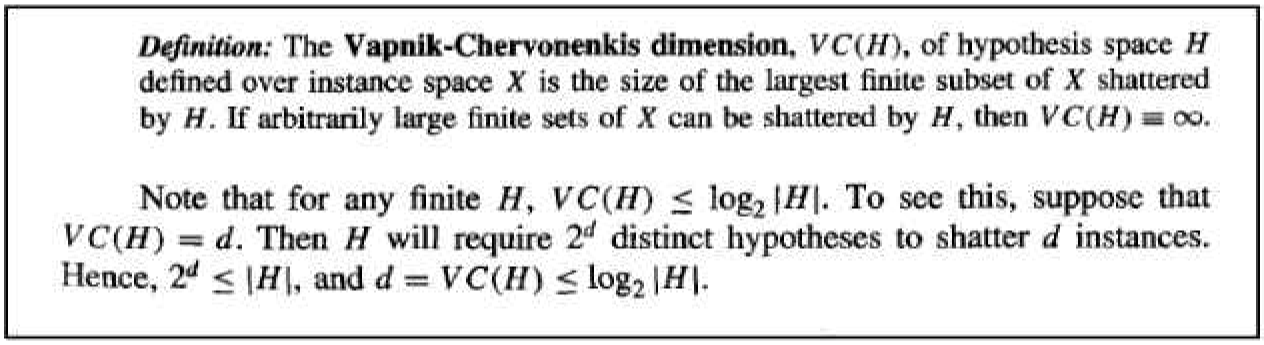
Hence H2 is not probably approximately correct

3.3 SAMPLE COMPLEXITY FOR INFINITE HYPOTHESIS SPACES

Answers : Vapink Shervonenkis(VC) Dimension

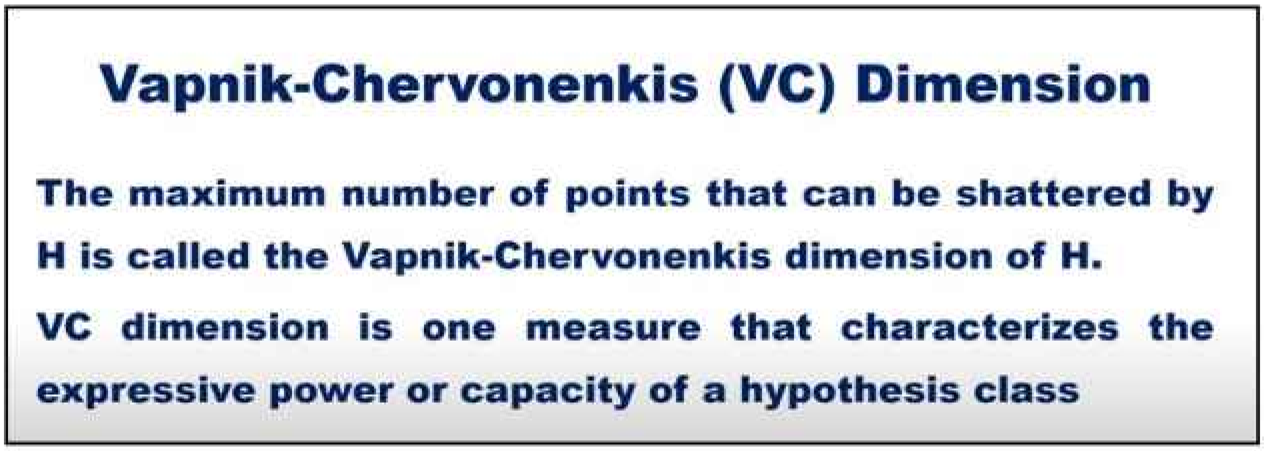
3.4 Vapink Shervonenkis(VC) Dimension

Definition of Vapink Shervonenkis(VC) Dimension: The VC dimension of the hypothesis space H, VC(H), is the size of the largest finite subset of the instance space X that can be shattered by H. If arbitrarily large finite subsets of X can be shattered by X then VC(H) = ∞



Definition: The **Vapnik-Chervonenkis dimension**, $VC(H)$, of hypothesis space H defined over instance space X is the size of the largest finite subset of X shattered by H. If arbitrarily large finite sets of X can be shattered by H, then $VC(H) \equiv \infty$.

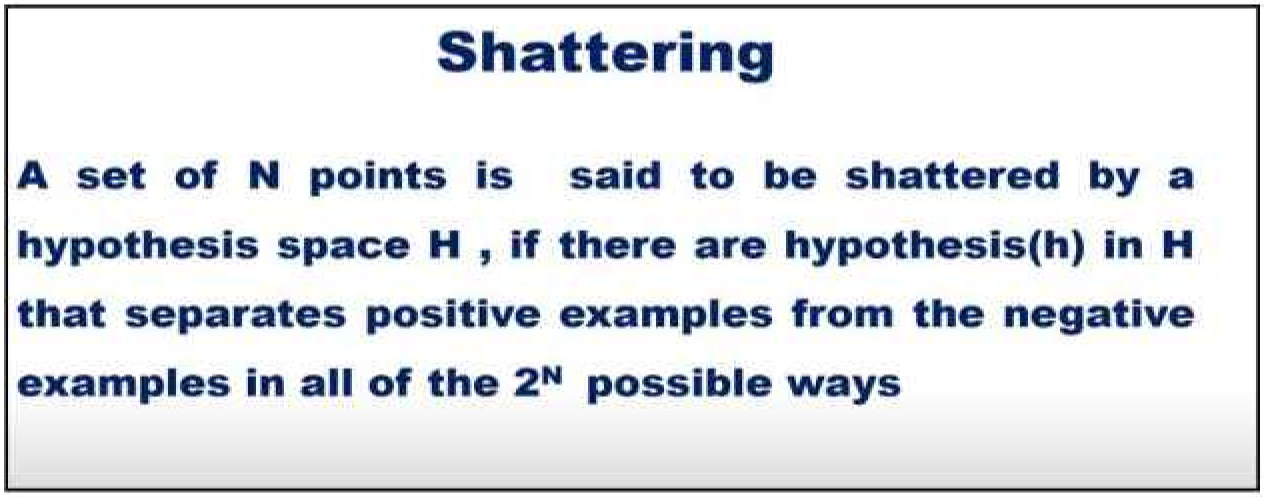
Note that for any finite H, $VC(H) \leq \log_2 |H|$. To see this, suppose that $VC(H) = d$. Then H will require 2^d distinct hypotheses to shatter d instances. Hence, $2^d \leq |H|$, and $d = VC(H) \leq \log_2 |H|$.



Vapnik-Chervonenkis (VC) Dimension

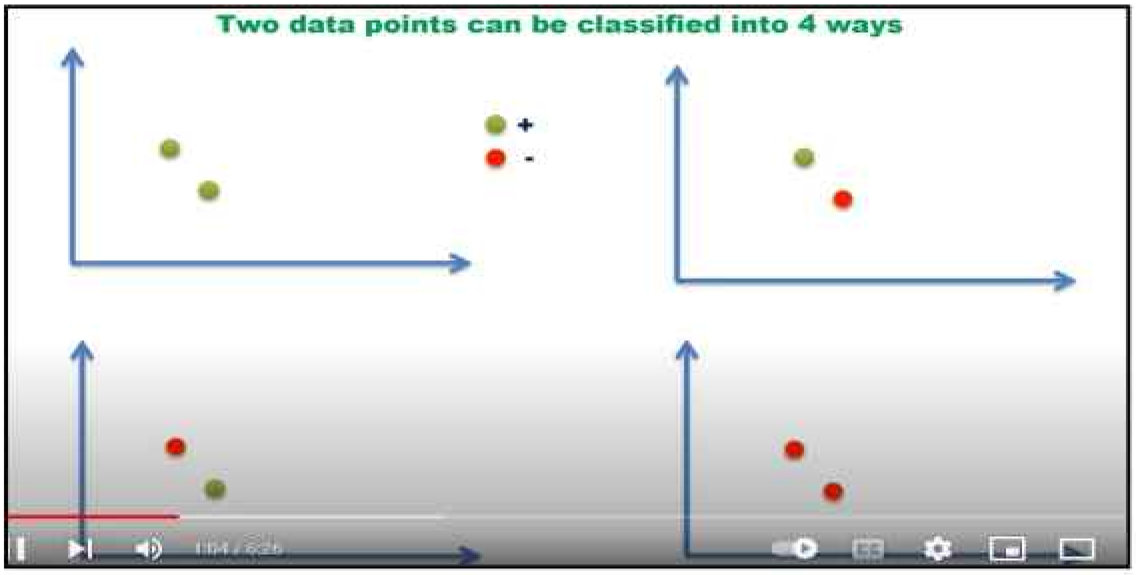
The maximum number of points that can be shattered by H is called the Vapnik-Chervonenkis dimension of H.

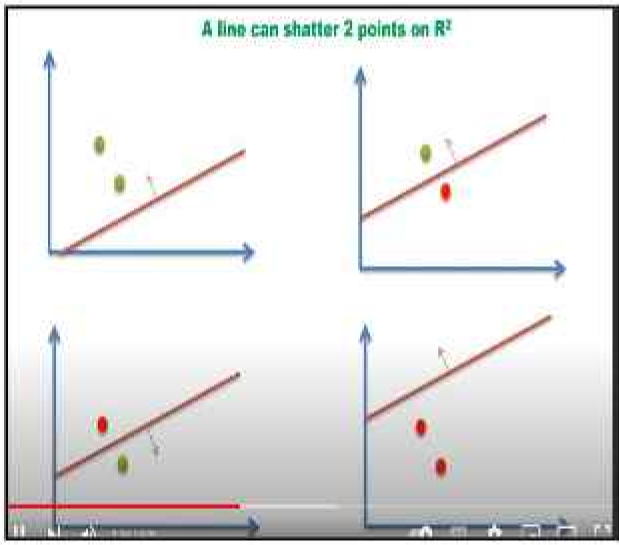
VC dimension is one measure that characterizes the expressive power or capacity of a hypothesis class

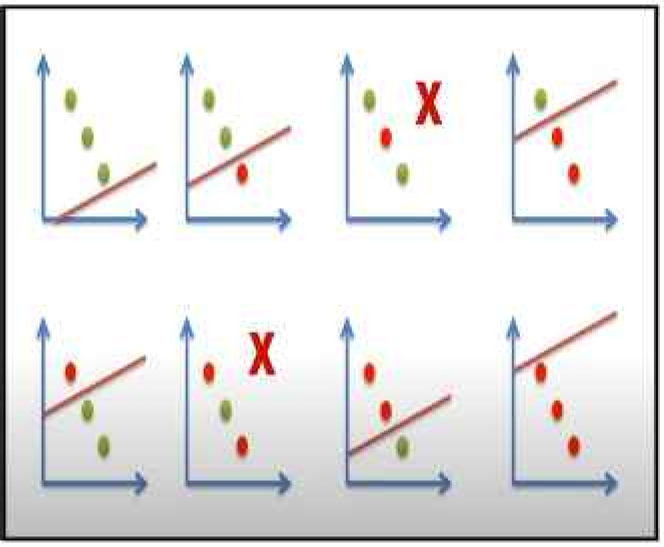


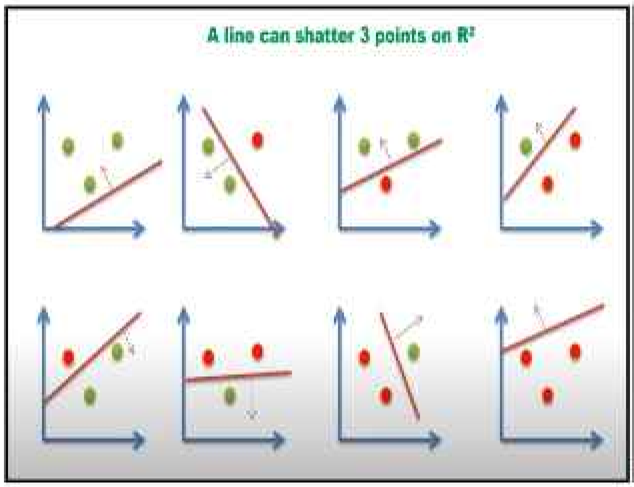
Shattering

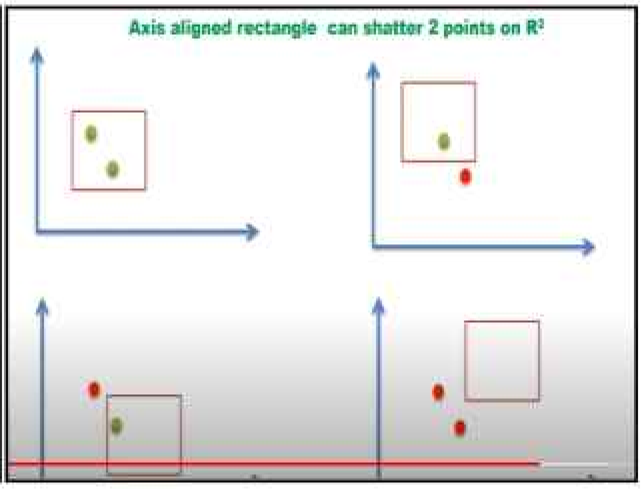
A set of N points is said to be shattered by a hypothesis space H , if there are hypothesis(h) in H that separates positive examples from the negative examples in all of the 2^N possible ways


Two data points can be classified into 4 ways
+
-


A line can shatter 2 points on R²


X
X


A line can shatter 3 points on R²


Axis aligned rectangle can shatter 2 points on R²

VCD (Axis Aligned Rectangle in R^2)= 4

3.5 Rule Learning

The Rule Based Learning represents knowledge in the form of IF-THEN rules that proves useful in Artificial Intelligence(AI) system . This type of learning is most suitable for analyzing data contains a mixture of numerical and qualitative attributed. Rule mining is useful and easy to understand by humans. The main aim of this learning is to discover interesting relations between variables and patents in large data sets. Several algorithms have proven useful which automatically induce rules for data to build more accurate AI systems .

Format of Rule:

$$\boxed{A \rightarrow B}$$

LHS A is called Antecedent and RHS B is called Consequent OR

A is called Condition and B is called Conclusion or

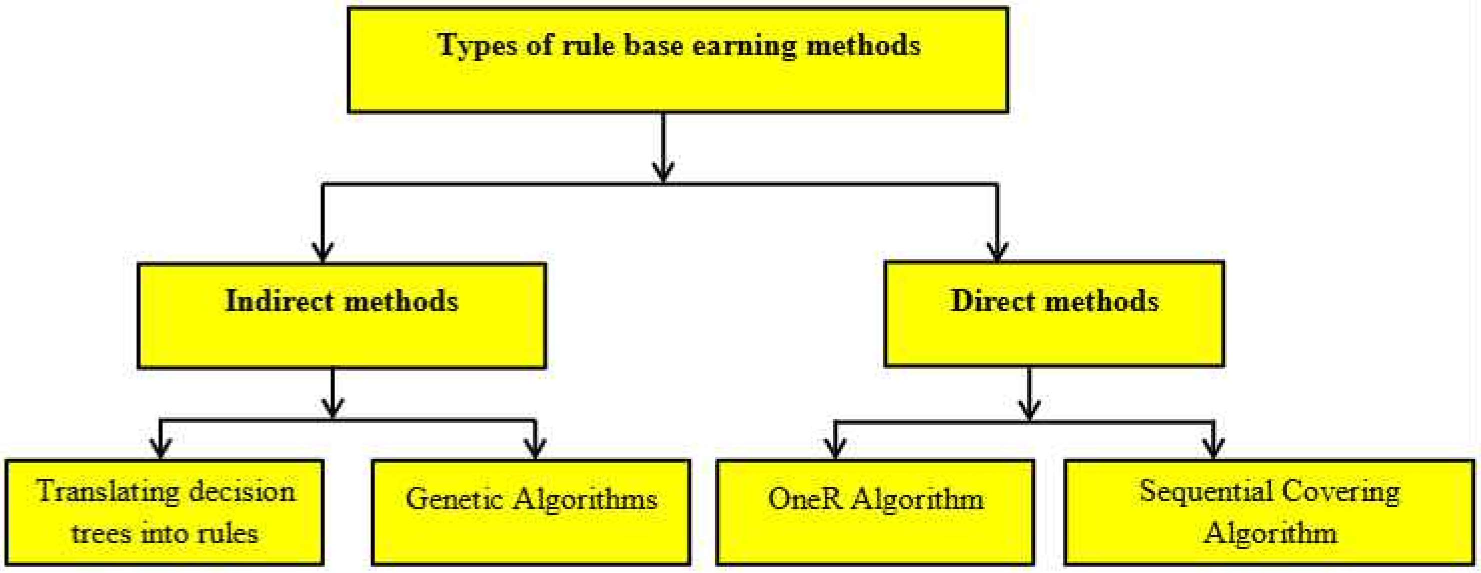
A is called body and B is called head or

A is conjunction of attribute-value pairs and B is class label (Target class, class prediction that is Yes or No)

Evolution Metrics of Rule: Two evolution metrics for rule , accuracy and coverage are defined as follows

$$\textbf{Accuracy} = \frac{\text{Number of instances that satisfy both antecedent and consequent of a rule}}{\text{instances that satisfy the antecedent of a rule.}}$$

$$\textbf{Coverage} = \frac{\text{Number of instances that satisfy antecedent of a rule}}{\text{Total number of instances in training data set}}$$

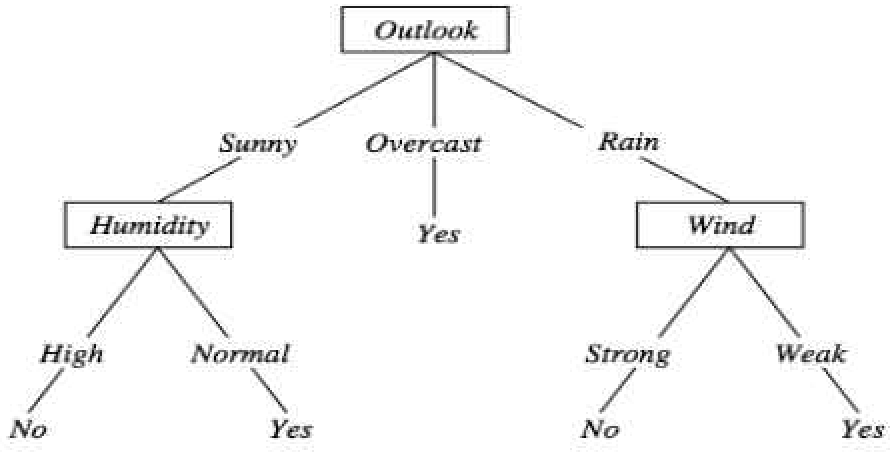

Types of rule base earning methods
Indirect methods
Direct methods
Translating decision trees into rules
Genetic Algorithms
OneR Algorithm
Sequential Covering Algorithm

Indirect methods: In this method, the rules are extracted from other classification models, the Decision Tree and Genetic Algorithms belongs to this category

Direct mothed: In this method, the rules are extracted directly from taring data. Some of examples are OneR Algorithm and Sequential Covering Algorithm

3.6 Translating decision trees into rules

In this method first decision is constructed from training data then it translated to rules.
Example decision tree for Playtennis as follows



The following three are rules predict class label as Yes

R1: IF (Outlook=Sunny $\wedge$ Humidity=Normal) THEN PlayTennis=Yes

R2: IF (Outlook=Overcast) THEN PlayTennis=Yes

R3: IF (Outlook=Rain $\wedge$ Wind=Week) THEN PlayTennis=Yes

The following two are rules predict class label as No

R4: IF (Outlook=Sunny $\wedge$ Humidity=High) THEN PlayTennis=No

R5: IF (Outlook=Rain $\wedge$ Wind=Strong) THEN PlayTennis=No

3.7 Sequential Covering Algorithms (Separate and Conquer Approach)

Sequential Covering is a popular algorithm based on Rule-Based Classification used for learning a disjunctive set of rules. The basic idea here is to learn one rule, remove the data that it covers, then repeat the same process. In this process, in this way, it covers all the rules involved with it in a sequential manner during the training phase. Sequential Covering Algorithms are based Separate and Conquer Approach

Some of example for Sequential Covering Algorithms

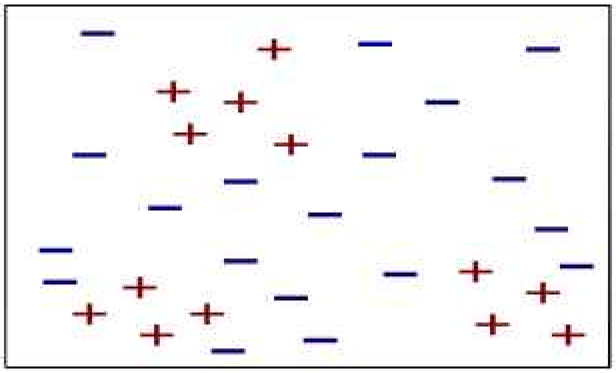
- AQ
- CN2
- RIPPER(Repeated Incremental Pruning to Produce Error Reduction
- PRISM
- M5
- FOIL(First Order Inductive Learner) and so on

In the Sequential Covering Algorithms steps as follows

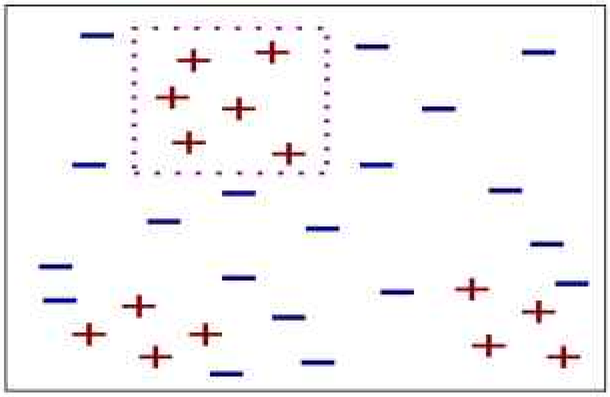
- Start from an empty rule
- Grow a rule using some Learn-One-Rule function
- Remove training records covered by the rule
- Repeat Step (2) and (3) until stopping criterion is met

Example 1: Consider the following Training Dataset:

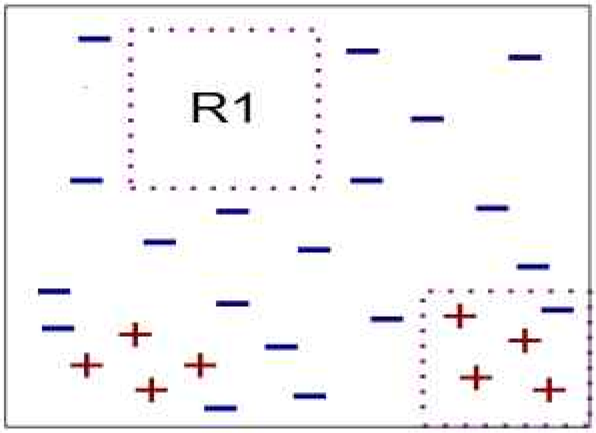
- Select rule from Training Dataset that covers most of positive examples and add this new rule to rule set
- Update the training dataset **by** remove those positive examples that have been covered selected rule from training dataset.
- Repeat this process until training dataset doesn't contain positive examples (Finally it contains only negative examples) as shown in figures



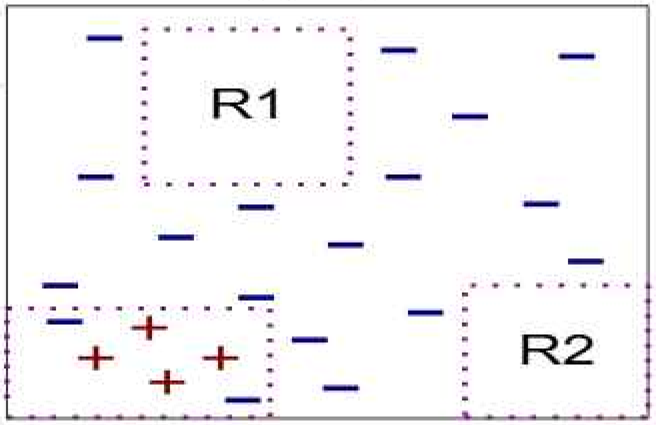
(i) Original Data



(ii) Step 1

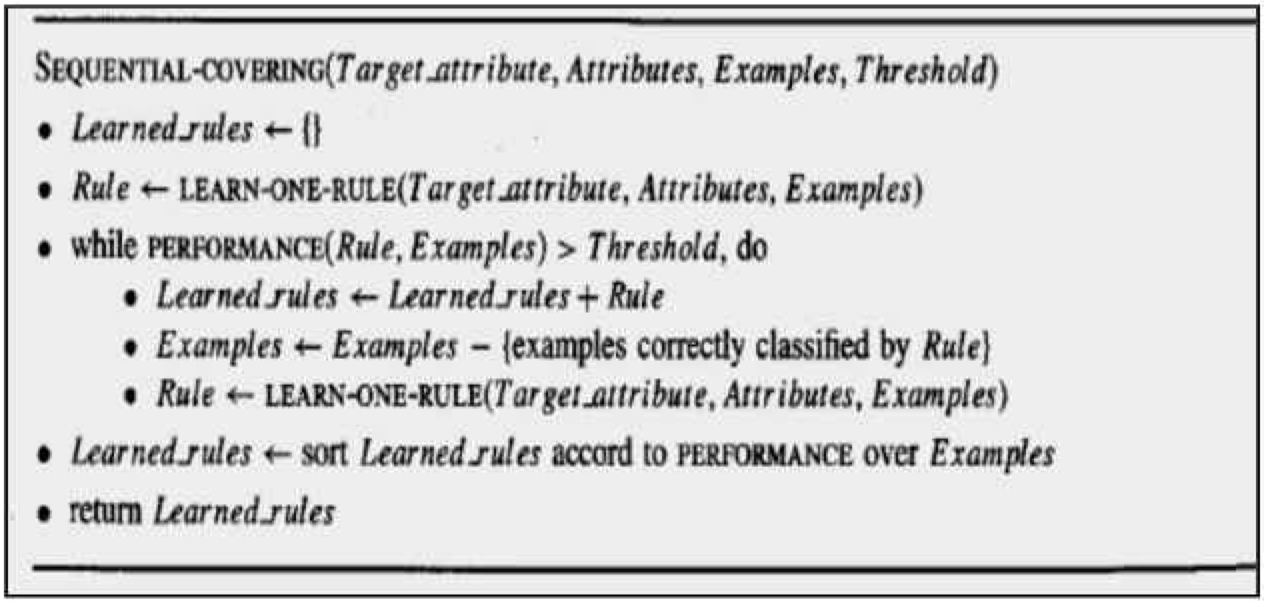


(iii) Step 2



(iv) Step 3

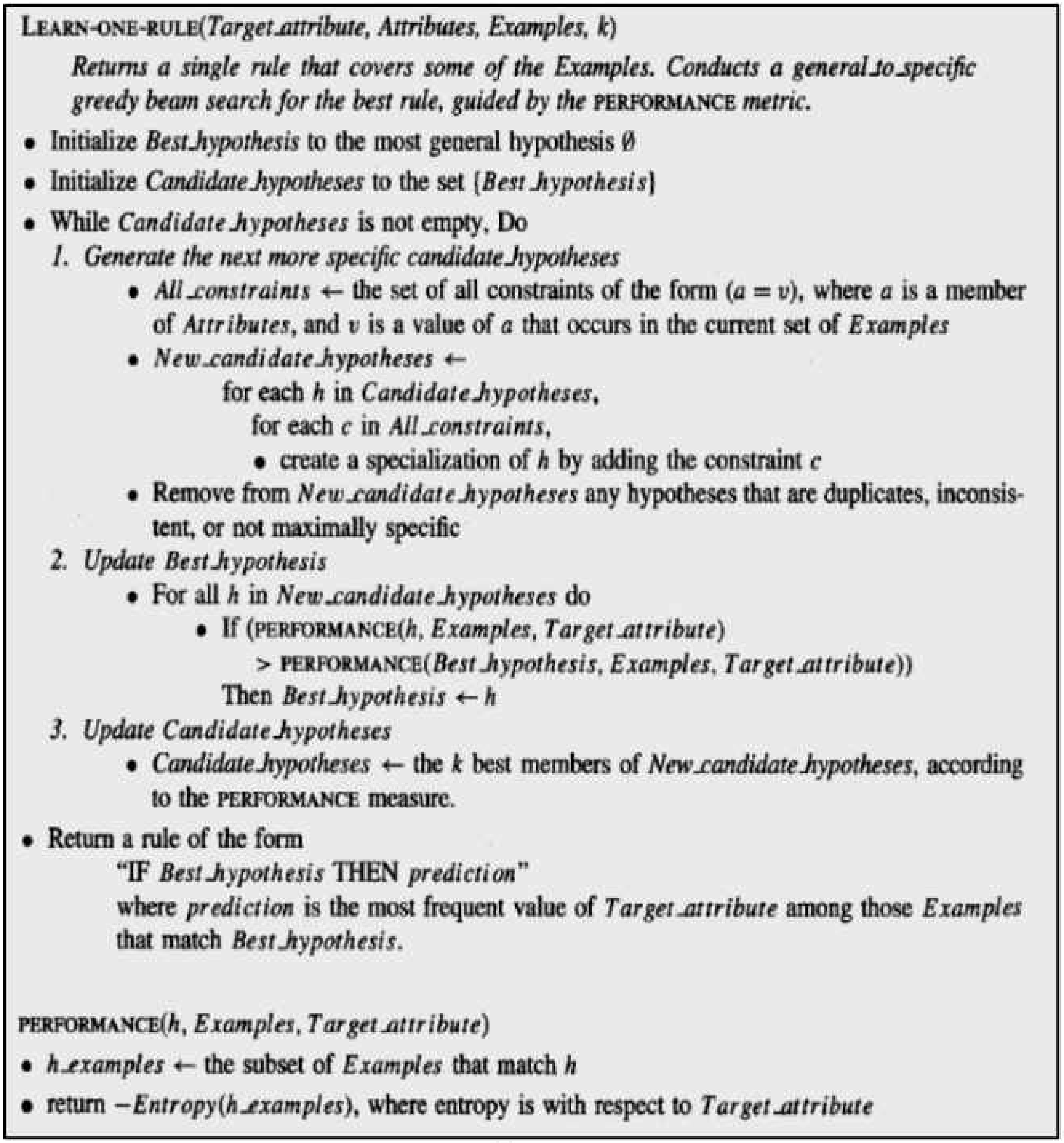
Sequential Covering Algorithms



```
SEQUENTIAL-COVERING(Target_attribute, Attributes, Examples, Threshold)
• Learned_rules ← {}
• Rule ← LEARN-ONE-RULE(Target_attribute, Attributes, Examples)
• while PERFORMANCE(Rule, Examples) > Threshold, do
    • Learned_rules ← Learned_rules + Rule
    • Examples ← Examples − {examples correctly classified by Rule}
    • Rule ← LEARN-ONE-RULE(Target_attribute, Attributes, Examples)
• Learned_rules ← sort Learned_rules accord to PERFORMANCE over Examples
• return Learned_rules
```

The sequential covering algorithm uses the subroutine (function) is called **LEARN-ONE-RELE**

LEARN-ONE-RELE subroutine (function) as follows



```
LEARN-ONE-RULE(Target_attribute, Attributes, Examples, k)
    Returns a single rule that covers some of the Examples. Conducts a general_to_specific
    greedy beam search for the best rule, guided by the PERFORMANCE metric.
• Initialize Best_hypothesis to the most general hypothesis $\emptyset$
• Initialize Candidate_hypotheses to the set {Best_hypothesis}
• While Candidate_hypotheses is not empty, Do
   1. Generate the next more specific candidate_hypotheses
        • All_constraints ← the set of all constraints of the form $(a = v)$, where $a$ is a member
          of Attributes, and $v$ is a value of $a$ that occurs in the current set of Examples
        • New_candidate_hypotheses ←
                  for each h in Candidate_hypotheses,
                     for each c in All_constraints,
                        • create a specialization of h by adding the constraint c
        • Remove from New_candidate_hypotheses any hypotheses that are duplicates, inconsis-
          tent, or not maximally specific
   2. Update Best_hypothesis
        • For all h in New_candidate_hypotheses do
            • If (PERFORMANCE(h, Examples, Target_attribute)
                 > PERFORMANCE(Best_hypothesis, Examples, Target_attribute))
              Then Best_hypothesis ← h
   3. Update Candidate_hypotheses
        • Candidate_hypotheses ← the k best members of New_candidate_hypotheses, according
          to the PERFORMANCE measure.
• Return a rule of the form
        "IF Best_hypothesis THEN prediction"
        where prediction is the most frequent value of Target_attribute among those Examples
        that match Best_hypothesis.


PERFORMANCE(h, Examples, Target_attribute)
• h_examples ← the subset of Examples that match h
• return −Entropy(h_examples), where entropy is with respect to Target_attribute
```

3.8 Learning First Order Rules

Lecture Outline:
• Why Learn First Order Rules?
• First Order Logic: Terminology
• The FOIL Algorithm

Propositional logic allows the expression of individual propositions and their truth-functional combination.

- E.g. propositions like *Tom is a man* or *All men are mortal* may be represented by single proposition letters such as P or Q (so, proposition letters may be viewed as variables which range over propositions)
- Truth functional combinations are built up using connectives, such as $\wedge$, $\vee$, $\neg$, $\rightarrow$– e.g. $P \wedge Q$
  - Inference rules are defined over propositional forms – e.g.

$$\frac{\begin{array}{c} P \rightarrow Q \\ P \end{array}}{Q}$$

–Note that if P is *Tom is a man* and Q is *All men are mortal*, then the inference that *Tom is mortal* does **not** follow in propositional logic

First order logic allows the expression of propositions and their truth functional combination, but it also allows us to represent propositions as assertions of predicates about individuals or sets of individuals

Example : propositions like *Tom is a man* or *All men are mortal* may be represented by Predicate-argument representations such as *man (tom)* or $\forall x\ (man(x) \rightarrow mortal(x))$
(So, variables range over individuals)

Inference rules permit conclusions to be drawn about sets/individuals – e.g. *mortal (tom)*

- First order logic is much more *expressive* than propositional logic – i.e. it allows a finer-grain of specification and reasoning when representing knowledge
- In the context of machine learning, consider learning the **relational** concept *daughter*(x, y) defined over pairs of persons x, y, where

- persons are represented by attributes: *(Name,Mother,Father,Male,Female)*

- *Training examples then have the form: (person1, person2, target attribute value)*

  E.g. (Name1 = Ann,Mother1 = Sue,Father1 = Bob,Male1 = F,Female1 = T)

  Name2 = Bob,Mother2 = Gill,Father2 = Joe,Male2 = T,Female2 = F)

  Daughter1,2 = Tī

: From such examples, a propositional rule learner such as ID3 or CN2 can only learn rules like:

IF $(Father1 = Bob) \wedge (Name2 = Bob) \wedge (Female1 = T)$

THEN $Daughter1,2 = T$

First Order Logic: Terminology

All expressions in first order logic are composed of:

- constants – e.g. *bob*, 23, *a*
- variables – e.g. *X,Y,Z*
- predicate symbols – e.g. *f emale, f ather* – predicates take on the values *True* or *False* only
- function symbols – e.g. *age* – functions can take on any constant as a value
- connectives – e.g. ∧, ∨, ¬, →(or←)
- quantifiers – e.g. ∀, ∃

A term is

- any constant – e.g. *bob*
- any variable – e.g *X*
- any function applied to any term – e.g. *age*(*bob*)

A **literal** is any predicate or negated predicate applied to any terms – e.g. female(sue), ¬father(X,Y)

– A ground literal is a literal that contains no variables – e.g. female(sue)

– A positive literal is a literal that does not contain a negated predicate – e.g. female(sue)

– A negative literal is a literal that contains a negated predicate – e.g ¬father(X,Y)

A **clause** is any disjunction of literals $L1 \vee \cdots \vee Ln$ whose variables are universally quantified (With wide scope)

• A **Horn clause** is any clause containing exactly one positive literal:

$H \vee \neg L1 \vee \cdots \vee \neg Ln$

Since $\neg L1 \vee \cdots \vee \neg Ln \equiv \neg (L1 \wedge \cdots \wedge Ln)$

and $(A \vee \neg B) \equiv (A \leftarrow B)$ (read $A \leftarrow B$ as "if B then A")

then a Horn clause can be equivalently written:

$H \leftarrow L1 \wedge \cdots \wedge Ln$

Note: the equivalent form in Prolog: H :- L1, ..., Ln.

• A **substitution** is a function $\theta = \{x1/t1, ..., xn/tn\}$ which when applied to an expression C yields a new expression C' with each variable xi in C replaced with term ti.

- C denotes the result of applying θ to C.
- A unifying substitution for two expressions $C1$ and $C2$ is any substitution q such that
- $C1\ \theta = C2\ \theta$

3.9 First Order Rule Inductive Learning (FOIL)

• FOIL was proposed by Quinlan, 1990, it is the natural extension of SEQUENTIAL-COVERING and LEARN-ONE-RULE to first order rule learning.

FOIL learns first order rules which are similar to Horn clauses with two exceptions:

- literals may not contain function symbols (reduces complexity of hypothesis space
- literals in body of clause may be negated (hence, more expressive than Horn clauses
- Like SEQUENTIAL-COVERING, FOIL learns one rule at time and removes positive examples covered by the learned rule before attempting to learn a further rule.

Unlike SEQUENTIAL-COVERING and LEARN-ONE-RULE, FOIL

- only tries to learn rules that predict when the target literal is true propositional version sought rules that predicted both true and false values of target attribute
- performs a simple hill-climbing search (beam search of width one)

FOIL searches its hypothesis space via two nested loops:

• The **outer loop** at each iteration adds a new rule to an overall disjunctive hypothesis (i.e. $rule1 \vee rule2 \vee ...$)

This loop may be viewed as a specific-to-general search

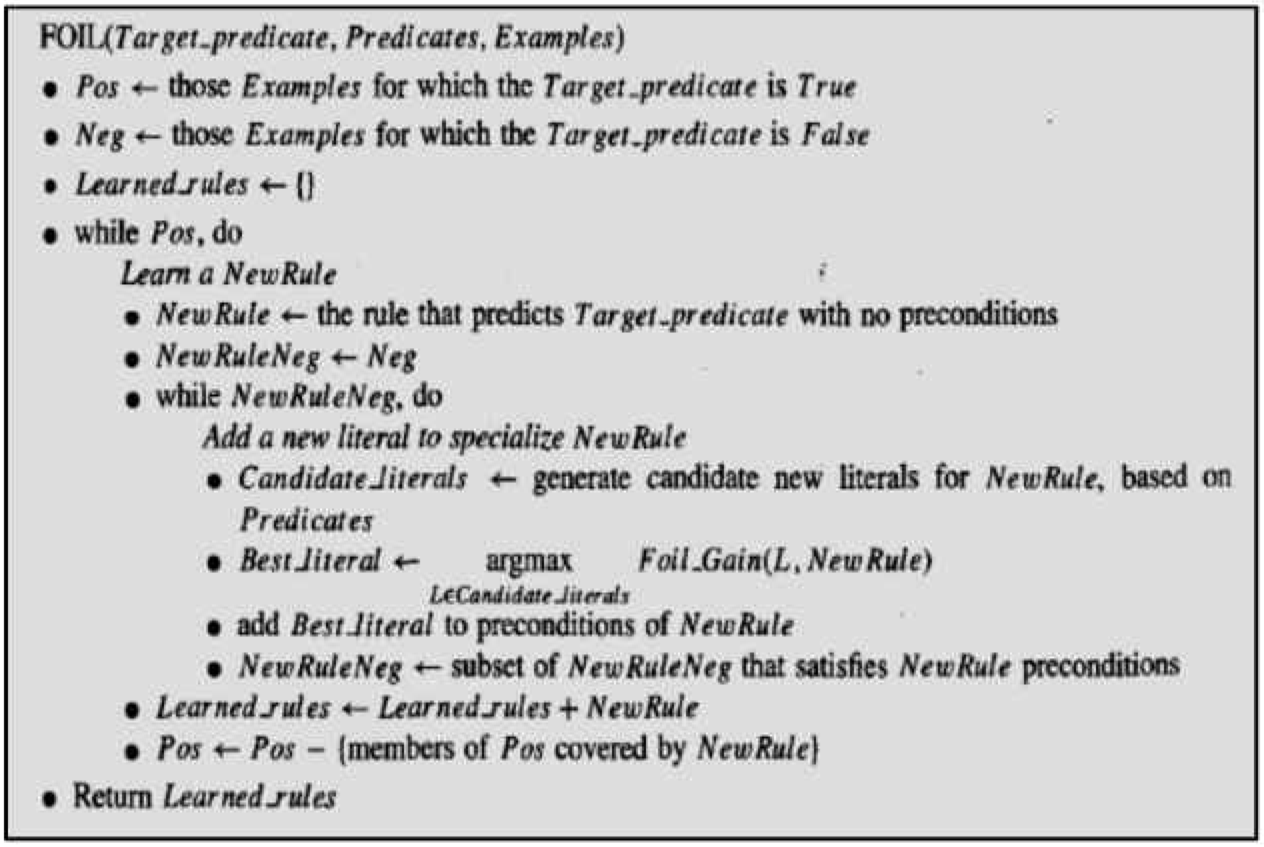
- starting with the empty disjunctive hypothesis which covers no positive instances
- stopping when the hypothesis is general enough to cover all positive examples

• The **inner loop** works out the detail of each specific rule, adding conjunctive constraints to the rule precondition on each iteration.

This loop may be viewed as a general-to-specific search

- starting with the most general precondition (empty)
- stopping when the hypothesis is specific enough to exclude all negative examples

3.9.1 FOIL Algorithm



```
FOIL(Target_predicate, Predicates, Examples)
• Pos ← those Examples for which the Target_predicate is True
• Neg ← those Examples for which the Target_predicate is False
• Learned_rules ← {}
• while Pos, do
      Learn a NewRule
      • NewRule ← the rule that predicts Target_predicate with no preconditions
      • NewRuleNeg ← Neg
      • while NewRuleNeg, do
            Add a new literal to specialize NewRule
            • Candidate_literals ← generate candidate new literals for NewRule, based on
              Predicates
            • Best_literal ← $\underset{L \in Candidate\_literals}{\operatorname{argmax}}$ Foil_Gain(L, NewRule)
            • add Best_literal to preconditions of NewRule
            • NewRuleNeg ← subset of NewRuleNeg that satisfies NewRule preconditions
      • Learned_rules ← Learned_rules + NewRule
      • Pos ← Pos − {members of Pos covered by NewRule}
• Return Learned_rules
```

3.9.2 FOIL: Explanation

The principal differences between FOIL and SEQUENTIAL-COVERING + LEARN-ONE-RULE are:

• In its inner loop search to generate each new rule, FOIL needs to cope with variables in the rule preconditions

• The performance measure used in FOIL is not the entropy measure used in LEARN-ONE-RULE since

- the performances of distinct bindings of rule variables need to be distinguished
- FOIL only tries to discover rules that cover positive examples

3.9.3 FOIL: Specializing the Current Rule

• Suppose we are learning a rule of the form: $P(x1, x2, \ldots, xk) \leftarrow L1 \ldots Ln$

• Then candidate specializations add a new literal of the form:

- *Q*(*v*1, . . . , *vr*), where
    - *Q* is any predicate in the rule or training data;
    - at least one of the *vi* in the created literal must already exist as a variable in the rule
- *Equal*(*x j*, *xk*), where *x j* and *xk* are variables already present in the rule; or
- The negation of either of the above forms of literals

4.9.4 FOIL: Performance Evaluation Measure

How do we decide which is the best literal to add when specializing a rule?

• To do this FOIL considers each possible binding of variables in the candidate rule specialization to constants in the training examples.

• For example, suppose we have the training data:

granddaughter(bill, joan) father(joan, joe) father(tom, joe)

female(joan) father(joe,bill)

and we also assume ("closed world assumption") that any literals

– involving predicates *granddaughter*, *father*, and *female*

– involving constants *bill*, *joan*, *joe*, and *tom*

– not in the training data

are false. E.g. ¬*granddaughter(bill, tom)* is also true.

• Given the initial rule: *granddaughter(X,Y)*←

FOIL considers all possible bindings of *X* and *Y* to the constants *bill*, *joan*, *joe*, and *tom*. Note that only {*X*/*bill*,*Y*/ *joan*} is a positive binding (i.e. corresponds to a positive training example). The other 15 bindings of constants to *X* and *Y* are negative.

3.9.5 Foil-Gain

• At each stage of rule specialization, candidate specializations are preferred according to whether they possess more positive and fewer negative bindings.

• The precise evaluation measure used by FOIL is:

$$Foil\_Gain(L,R) = t\left(\log_2 \frac{p_1}{p_1+n_1} - \log_2 \frac{p_0}{p_0+n_0}\right)$$

Where

- L is the candidate literal to add to rule R
- p_0 = number of positive bindings of R
- n_0 = number of negative bindings of R
- p_1 = number of positive bindings of $R+L$
- n_1 = number of negative bindings of $R+L$
- t is the number of positive bindings of R also covered by $R+L$

3.9.6 Summary/Observations of FOIL

FOIL extends the SEQUENTIAL-COVERING and LEARN-ONE-RULE algorithms for propositional rule learning to first order rule learning

• FOIL learns in two phases:

- an outer loop which acquires a disjunction of Horn clause-like rules which together cover the positive examples
- an inner loop which constructs individual rules by progressive specialization of a rule through adding new literals selected according to the FOIL-gain measure until no negative examples are covered

3. 10 Induction as Inverted Deduction

A second, quite different approach to inductive logic programming is based on the simple observation that induction is just the inverse of deduction! In general, machine learning involves building theories that explain the observed data. Given some data D and some partial background knowledge B, learning can be described as generating a hypothesis h that, together with B, explains D. Put more precisely, assume as usual that the training data D is a set of training examples, each of the form $(x_i, f(x_i)\}$. Here x_i denotes the *ith* training instance and $f(x_i)$ denotes its target value. Then learning is the problem of discovering a hypothesis h, such that the classification $f(x_i)$ of each training instance x_i follows deductively from the hypothesis h, the description of x_i and any other background knowledge B known to the system.

Induction is finding h such that

$$\boxed{(\forall (x_i, f(x_i)) \in D)\ B \wedge h \wedge x_i \vdash f(x_i)}$$

where
• x_i is the ith training instance
• $f(x_i)$ is the target function value for x_i
• B is other background knowledge.

The expression $X \vdash Y$ is read "Y follows deductively from X," or alternatively "X entails Y." Expression (10.2) describes the constraint that must be satisfied by the learned hypothesis h; namely, for every training instance x_i, the target classification $f(x_i)$ must follow deductively from B, h, and x_i

As an example, consider the case where the target concept to be learned is "pairs of people (u, v} such that the child of u is v," represented by the predicate Child(u, v).

Assume we are given a single positive example Child (Bob, Sharon), where the instance is described by the literals Male(Bob), Female(Sharon), and Father(Sharon, Bob).

Furthermore, suppose we have the general background knowledge Parent(u, v) $\leftarrow$ Father(u, v).

We can describe above equation as follows
x_i Male(Bob), Female(Sharon), Father(Sharon, Bob)

$f(x_i)$: Child(Bob, Sharon)

B: Parent(u, v) $\leftarrow$Father(u, v)

Inverse resolution

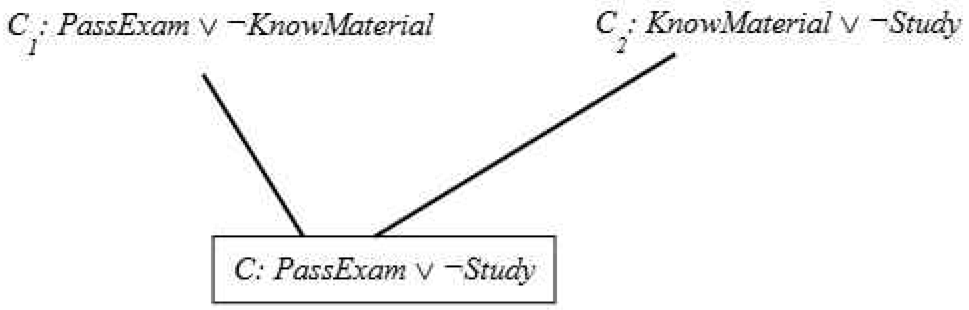
3. 11 Inverse resolution

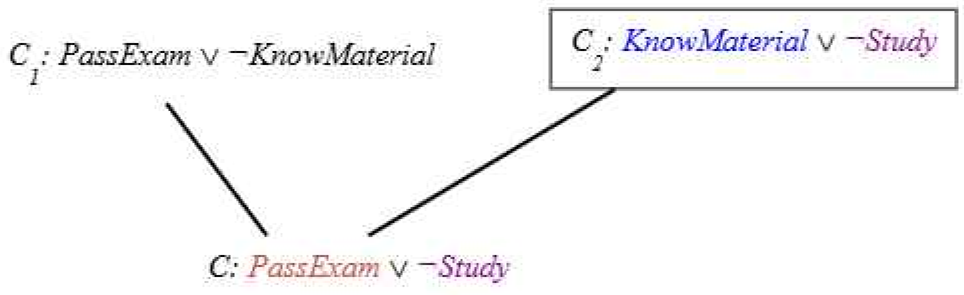
It is easiest to introduce the resolution rule in propositional form, though it is readily extended to first-order representations. Let L be an arbitrary propositional literal, and let P and R be arbitrary propositional clauses. The resolution rule is

$$\frac{\begin{array}{c} P \vee L \\ \neg L \vee R \end{array}}{P \vee R}$$

1. Given initial clauses *C1* and *C2*, find a literal *L* from clause *C1* such that $\neg L$ occurs in clause *C2*.
2. Form the resolving *C* by including all literals from *C1* and *C2*, except for *L* and $\neg L$. More precisely, the set of literals occurring in the conclusion *C* is

$$C = (C1 - \{L\}) \cup (C2 - \{\neg L\})$$





Inverted Resolution (Propositional)

Given initial clauses C_1 and C, find a literal L that occurs in clause C_1, but not in clause *C*.

2. Form the second clause C_2 by including the following literals

$C_2 = (C - (C_1 - \{L\})) \cup \{\neg L$

First Order Resolution

1. Find a literal L_1 from clause C_1 , literal L_2 from clause C_2, and substitution θ such that

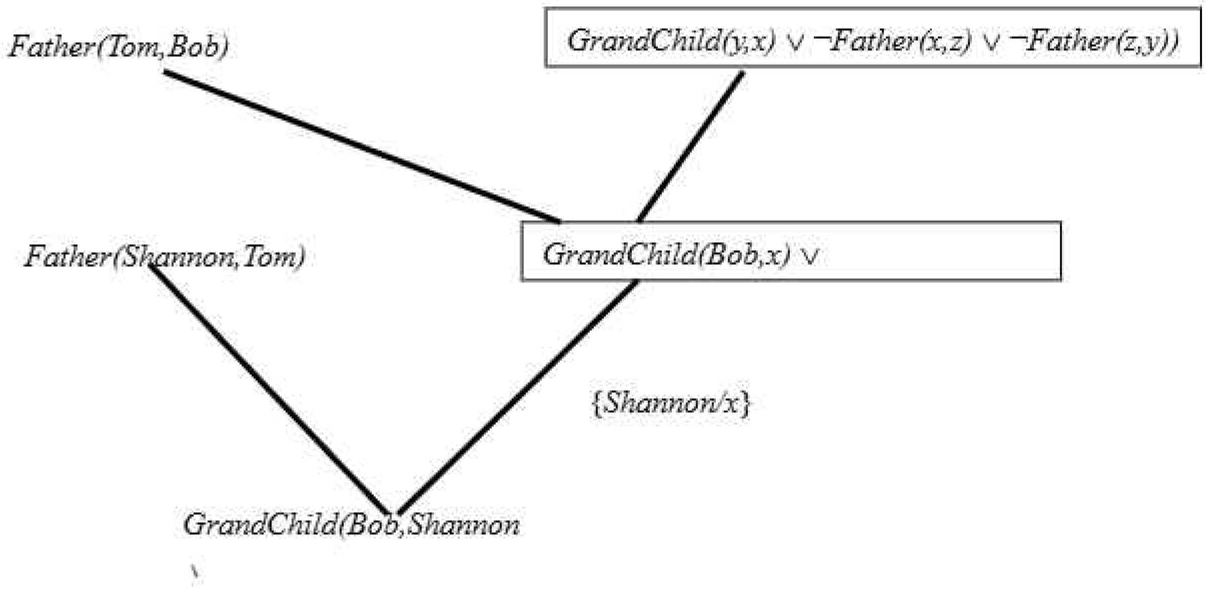
$$L_1\theta = \neg L_2\theta$$

2. Form the resolvent *C* by including all literals from $C_1\theta$ and $C_2\theta$, except for L_1 theta and $\neg L_2\theta$. More precisely, the set of literals occuring in the conclusion is

$$C = (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta$$

Inverting:

$$C_2 = (C - (C_1 - \{L_1\})\,\theta_1)\,\theta_2^{-1} \cup \{\neg L_1 \theta_1 \theta_2^{-1}\}$$



3. 12 PROGOL

PROGOL: Reduce combinatorial explosion by generating the most specific acceptable h

1. User specifies H by stating predicates, functions, and forms of arguments allowed for each
2. PROGOL uses sequential covering algorithm.

    For each $(x_i, f(x_i))$

    Find most specific hypothesis h_i s.t.

    $B \wedge h_i \wedge x_i \vdash f(x_i)$

    actually, only considers k-step entailment

3. Conduct general-to-specific search bounded by specific hypothesis h_i, choosing hypothesis with minimum description length