ML UNIT-1 IMPORTANT QUESTIONS

8) EXPLAIN ABOUT INDUCTIVE BIAS IN DECISION TREE LEARNING.

Ans:-

INDUCTIVE BIAS IN DECISION TREE LEARNING:

→ An Inductive Bias is a set of assumptions, which along with training data, deductively justify the classification assigned by the learner to future instances.

→ For a given collection of Training exs. there are many decision trees consistent with it. Using Inductive bias, we select one of these consistent hypotheses over others.

→ ID3 chooses, the 1st acceptable tree it encounters in simple-to-complex, Hill climbing search through the space of possible trees.

→ It aims to select shorter trees over longer & trees that place the attributes which have highest info-gain, closest to the root.

→ example, in BFS-ID3 (Breadth 1st search), it 1st considers all trees with depth 1 & later depth 2. It tries to search for decision tree consistent with training data & returns the smallest consistent tree at that search depth.

→ Restriction Biases & Preference Biases :-

ID3 search

* searches complete hypothesis space.
(one capable of expressing any finite discrete valued func.)
* Searches Incompletely through this space.
* searches from simple to complex hypotheses until a termination condition is met (i.e. it finds a hypothesis consistent with the data).
* Its Inductive bias is solely a consequence of hypotheses by its search strategy
* Its hypothesis space introduces no additional bias
* Inductive bias of ID3 follows from its search strategy
* Inductive bias of ID3 is a Preference for certain hypothesis over others. Thus it is called preference Bias or Search Bias.
* Adv:- preference bias is more desirable becoz it allows the learner to work within a complete hypothesis space i.e assumed to contain the unknown Target func.

Candidate elimination Algo.

* Searches Incomplete hypothesis space (one that can express only a subset of potentially teachable concepts)
* Searches Completely through this space.
* It finds every hypothesis consistent with the data (training)
* Its Inductive bias is solely a consequence of expressive power of its hypothesis rep".
* Its ~~hypothesis~~ Search strategy introduces no additional bias.
* Its Inductive bias follows from its def" of Search space
* Bias of this Algo. is in the form of a Categorical Restriction on the set of hypotheses considered.
Thus it is called Restriction Bias or language bias.
* DisAdv:- or undesirable
Its is undesirable becoz it strictly limits the set of potential hypotheses. It introduces the possibility of excluding unknown Target func. Altogether

→ Why prefer short Hypotheses?

Accdg to Occam's Razor, prefer the simplest hypothesis that fits the data.

- Because, there are fewer short hypotheses than long ones but it can be complex as well. and may fail to generalize correctly to subsequent data.

- Ex:- In Decision Tree hypotheses, Assume 20 training examples, A 5 node tree with a less likely statistical coincidence is prefered over 500 node hypothesis.

- Another problem with Occam's Razor, stmt. is the size of hypothesis. It is determined internally by the learner. Two learners with 2 diff. internal repns. can arrive at diff. hypotheses.

- Solution will be to have a common process of evolution that shall create an internal repn.

20) DESCRIBE IN DETAIL STEPS FOR DESIGNING A LEARNING SYSTEM.

Ans:-

2. Designing a Learning system:

- Before understanding design issues & design approaches, first, we shall understand, to design a problem
- Exampled used: GOAL: Entering world checkers tournament
  TASK: Learn to play checkers.
  PERFORMANCE MEASURE: % of games it wins in the world tournament

→ Choosing the Training Experience :-

- choose a type of training experience from which our sys will learn. Its v. imp for the success/failure of the learner.

feature

① it key feature to select it will be to see if training experience will provide direct/indirect feedback.

ex:- Direct - Individual checkers board states & correct move for each.

Indirect - Move seqs & final outcomes of various games played

- The disadv of this is "credit assignment"
- Learning from direct feedback is easier.

② Degree to which learner controls the seq. of training examples.

- ex: (i) learner dependent on teacher to ~~set~~ select informative boards & to provide correct move for each.

(ii) learner can propose board states & if finds a particular move confusing then it can ask teacher.

(iii) Learner can have control on both board states & training classifications. & there is no teacher present

③ How well it rep. distribution of examples over which the final sys. performance P must be measured:

- generally, if future test example & distribution of learning examples (train) are similar then learning is more reliable.

- Ex example of checker game, if training experience is E, i.e. games & played against itself & performance measure P, is % of games won in world tournament then there is a danger that training experience will not be similar to the situation where it will be tested in future.

- But for simplicity, we still assume a fully-specified Tasks:

(Task) T: playing checkers

(Performance Measure) P: % of games won in world Tournament

(Training experience) E: games played against itself

- Foll. steps are needed next to complete the design:
  (i) exact type of knowledge to be learned.
  (ii) Rep" for this Target knowledge
  (iii) learning mechanism.

→ choosing the Target function :-

- we shall now determine the Target knowledge & how will it be used.
- Ex; take a checkers playing prog. Assume, it can generate the "legal" moves from any board state.
- lets say, prog needs to learn only the "best" move among them
- Assume "ChooseMove" be a func. which chooses best move for any given board state.
- "ChooseMove : B → M" is the notation used to accept any board as i/p from a set of "legal board states" B and produces "M", which is an o/p move from a set of legal moves M. But this func is V. difficult to learn since it is an "indirect" training experience.
- Alternative Target func.: Let say, an evaluation func. 'V' assigns a numerical score to any board state.
  so V: B → R ⇒ V will map any legal board state from set B to some real value.
  ⇒ The better board states the greater shall be "R" value.
  This method is comparatively easier for the sys. to learn and select best move from any board state.

Example:- if b is a final board state i.e won ⇒ V(b) = 100
if b " " " " i.e lost ⇒ V(b) = -100
if b " " " " i.e Drawn ⇒ V(b) = 0
if b is not a final state in game ⇒ V(b) = V(b')
where b' = best final board state that can be achieved

→ Choosing a repⁿ for Target func :-

- We've got the ideal target func "V". Now we must choose $\hat{V}$ i.e function approximation
Because learning algo. can't acquire ideal or perfect target func "V". Rather, they acquire only some approxⁿ to the target func. so that's why we shall choose $\hat{V}$ now.
- For the case of checkers game $\hat{V}$ can be:
  (i) collection of rules that match against features of board state
  (ii) Large table with distinct entry specifying the value for each distinct board state.
- for expressing $\hat{V}$ which is more similar to V, one needs a large amount of training data.
- $\hat{V}$ will be calculated as a linear combⁿ of foll - board features
  (i) x_1 : no. of black pieces on board
  (ii) x_2 : " " red " "
  (iii) x_3 : " " black kings " "
  (iv) x_4 : " " Red kings " "
  (v) x_5 : no of black pieces threatened by red.
  (vi) x_6 : " " red " " " black.
- So, learning prog. is
$$\hat{V}(b) = w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$
where, $w_0, w_1 \text{ to } w_6$ = weights chosen be algo.
- weights determine the relative importance of various board features in finding out value of board
w_0 adds an additive constant to the board value.
After this step now, we have a partial design of checkers learning program.

Specification of learning Task:
① Task T: playing checkers
② performance measure P: % of games won in World tournament
③ Training Experience E: games played against itself

implementation of learning prog:
④ Target func.: V: Board → R
⑤ Target func Repn: $\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$

→ choosing a func. Approx^n Algo.:—

- For learning the Target func. $\hat{V}$, a set of training examples is needed, i.e $V_{train}(b)$.
- Each example will describe a specific board state "b" & training value for it i.e $V_{train}(b)$.
- Consider an example, where, black has won game. then $x_2 = 0$ ⇒ There are no red pieces on board & value of $V_{train}(b) = +100$.
→ So the ordered pair of training example will be in the form $\langle b, V_{train}(b) \rangle$.
For above example taken it is
$\langle x_1 = 3, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 0, +100 \rangle$

→ Estimating Training Values:—

The Only Training info. thats available is whether game was won/lost. So we can assign Training values v. easily.
- But, some times there are "Intermediate" board states that occurs before game ends i.e we cant say whether game path was good/bad or whether a subsequent poor move made the game to lose it.
→ There is a simple approach for such

- Intermediate board states:
"Assign training value of $V_{train}(b)$ for any intermediate board state 'b' to be $\hat{V}(Successor(b))$"
i.e $V_{train}(b) \leftarrow \hat{V}(Successor(b))$.

where $\hat{V}$ = Learner's current approx^n to V.
Successor(b) = next board state following b for which it is again the prog's turn to move.

→ Adjusting Weights:-

- We must choose weights w_i so that it fits best to the set of training examples $\{\langle b, V_{train}(b)\rangle\}$
- Best fit:- First define best hypothesis as a set of weights which minimises E (squared error) b/w training values & predicted values by $\hat{V}$.

$$\Rightarrow E \equiv \sum_{\langle b, V_{train}(b)\rangle \in \text{Training examples}} (V_{train}(b) - \hat{V}(b))^2 .$$

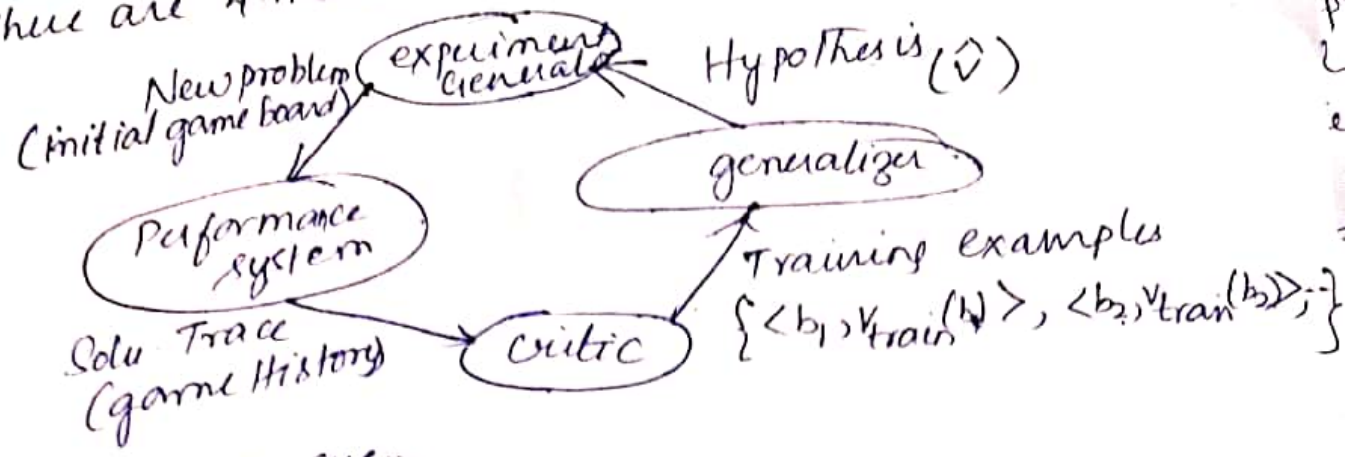
- There are many Algos. for finding weights of a linear func. that minimises E. One of them is LMS (Least Mean squares) Training Rule.

- LMS Weight update Rule:-
For each training example $\langle b, V_{train}(b)\rangle$
  - use current wt. to calculate $\hat{V}(b)$.
  - For each wt w_i, update

$$w_i \leftarrow w_i + \eta (V_{train}(b) - \hat{V}(b)) x_i$$

where η = small const (0.1), which moderates degree of wt update
if $V_{train}(b) - \hat{V}(b)$ = +ve ⇒ each wt is ↑ in prop. to value of its x_i
-ve ⇒ " " ↓ " "
0 ⇒ No wt. change.

→ The Final Design:-
There are 4 modules to describe the final Design viz:



(i) performance sys:-
- Solves given performance Task
- In this ex., it is playing the checkers game, i.e. Learned Target function.
- Takes new game as i/p & produces a part of its solu. i.e game History as o/p
- Strategy for performance sys:
  - Select its next move at each step determined by its evaluation func. $\hat{V}$.
  - As $\hat{V}$ ↑ses ⇒ performance ↑ses

(ii) Critic:-
- i/p: History / part of Trace of solu.
- o/p: Set of Training examples of the Target func.
- Each Training ex., is nothing but a game state b' & and its estimate of Target func. $V_{train}(b)$.

(iii) Generalizer:-
- i/p: Training examples
- o/p: Hypothesis $\hat{V}$ = estimate of Target func.
- It takes specific training examples & generalises a general func. (Hypothesis) that covers these examples.
- In this ex., it has followed LMS algo. It uses Learned weights $w_0, w_1, \ldots, w_6$ to describe the o/p hypothesis $\hat{V}$.

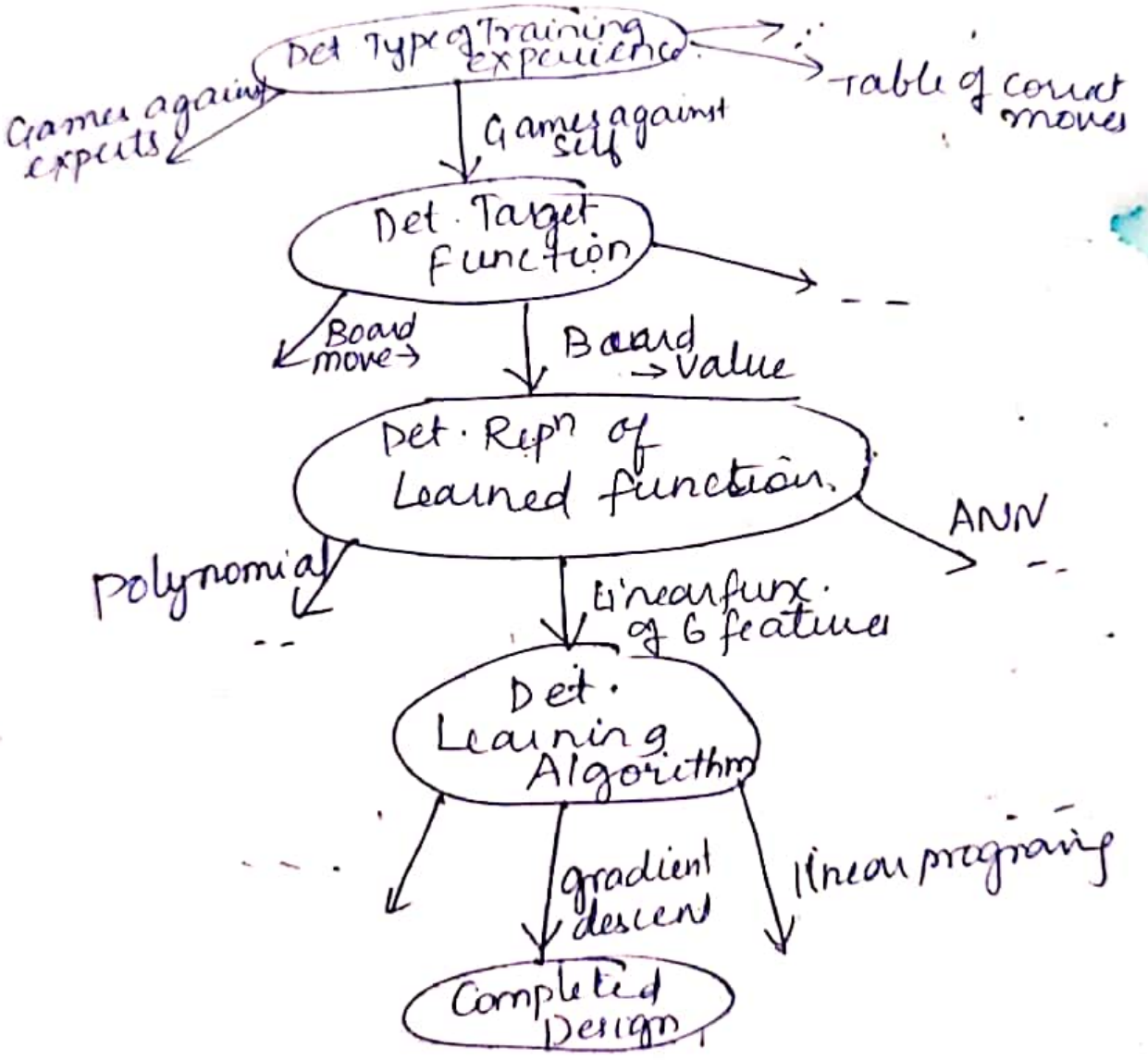
(iv) Experiment generator:-

i/p: current Hypothesis
o/p: new problem (i.e initial board state) for the performance sys. to explore.

Its role is to pick new practice problems that maximize learning rate of overall system.

→ The seq. of design choices made for the checkers prog. is summarised below:

Assumptions:- — Single linear evaluation fun.
— 6 specific board features



Q3) DISCUSS ABOUT FIND-S ALGORITHM IN FINDING A MAXIMUM SPECIFIC HYPOTHESIS.

Find-S: Finding a Maximally Specific Hypothesis:

→ It is an appln of more general_than partial ordering.

→ Here, we begin with most specific possible, hypothesis in H. & generalize it, when it fails to cover an observed +ve Training example. (classify as +ve)

→ Find-S Algorithm: -

1. Initialise h to most specific hypothesis in H.
2. For each +ve Training instance x:
   - For each attribute constraint a_i in h
     if Constraint a_i is satisfied by x
     then do nothing
     else
     Replace a_i in h with next more general constraint satisfied by x.
3. O/p hypothesis h.

→ Example:-

- consider "Enjoy sport task".

(i) Initialise h to most specific hypothesis in H.

$h \leftarrow \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$.

(ii) From table 2.1, none of constraints are satisfied by this so it is replaced by next more general constraint that fits this example.

$h \leftarrow \langle$Sunny, Warm, Normal, Strong, Warm, Same$\rangle$

[Its too specific bcoz all instances are -Ve except one +ve Traing example, (See Table 2.1 to relate)]

→ The second Training example forces algo. to further generalise h, which can be done by substituting any attribute value in 'h' by a "?".

$h \leftarrow$ <Sunny, warm, ?, Strong, warm, Same>

→ Upon encountering a -ve Training example (3rd one) Find_S algo simply ignore it. So no revision of "h" is needed.

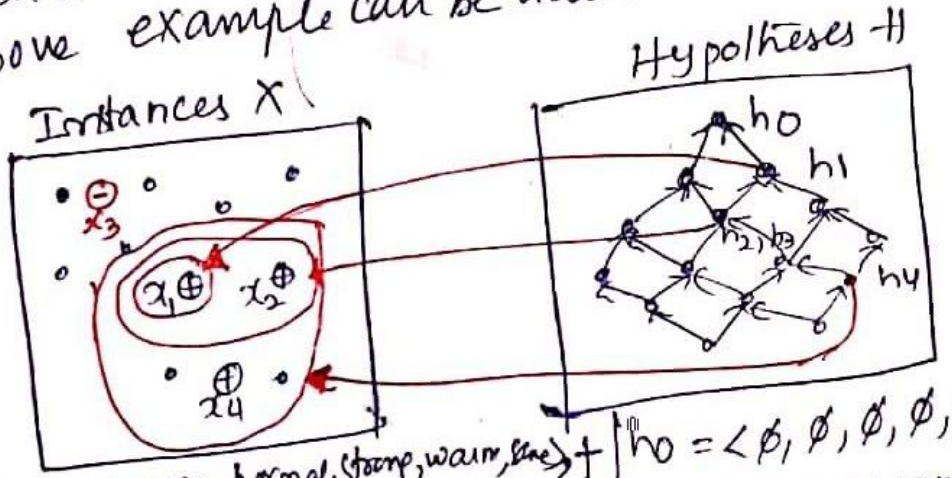
→ the 4th example (+ve) leads to further generalisation of "h".

$h \leftarrow$ <Sunny, warm, ?, strong, ?, ?>

→ In this Algo., "More_general_than" partial ordering is thus used to organise the search for an acceptable Hypothesis.

→ At each step, the hypothesis is generalised as far as necessary to cover the new +ve examples thus making the hypothesis more consistent with Training examples. Hence it is named as "Find-S" Algorithm

→ Above example can be illustrated further with foll. fig.



x_1 = <Sunny, warm, Normal, Strong, warm, Same>, +
x_2 = <Sunny, warm, high, Strong, warm, same>, +
x_3 = <rainy, cold, high, strong, warm, change>, −
x_4 = <Sunny, warm, High, strong, Cool, change>, +

h_0 = <∅, ∅, ∅, ∅, ∅, ∅>
h_1 = <Sunny, warm, Normal, strong, warm, same>
h_2 = <Sunny, warm, ?, strong, warm, same>
h_3 = <Sunny, warm, ?, strong, warm, same>
h_4 = <Sunny, warm, ?, strong, ?, ?>

→ "Find-S" is guaranteed to o/p most specific hypothesis within H that is consistant with +ve training example. Finally, it is also consistent with -ve examples provided the correct target concept is contained in H & also provided the training examples are correct.

→ Few voids in this Algo. in the form of Questions are:

* Has learner converged to correct target concept?
* Why prefer most specific hypothesis?
* Are Training examples consistent
* What if there are several maximally specific Consistent hypotheses.?

Q4. EXPLAIN IN DETAIL "CANDIDATE_ELIMINATION" LEARNING ALGORITHM.

A) → Candidate_Elimination Learning Algorithm:-

1. Initialise G to set of maximally general Hypotheses in H
2. " S " " " " specific " "
3. For each training example 'd', do

- If d is positive example
  - (i) Remove from 'G' any hypothesis inconsistent with 'd'.
  - (ii) For each hypothesis 's' in S that is inconsistent with 'd':
    - (a) Remove 's' from 'S'
    - (b) Add to 'S' all minimal generalizations 'h' of 's' such that
      - (i) h is consistent with d
      - (ii) some members of G is more general than h.
    - (c) Remove from S, any hypothesis i.e more general than another hypothesis in S.

- If d is Negative example:
  - (i) Remove from 'S' any hypothesis inconsistent with 'd'.
  - (ii) For each hypothesis 'g' in G i.e not consistent with 'd'
    - (a) Remove 'g' from 'G'.
    - (b) Add to 'G' all minimal generalizations 'h' of 'g' such that
      - (i) h is consistent with d
      - (ii) some members of S is more specific than h.
    - (c) Remove from G, any hypothesis i.e less general than another hypothesis in G.

Example:-

→ Initialise G & S

$G_0 \leftarrow \{<?,?,?,?,?,?>\}$ Most general hypothesis in H

$S_0 \leftarrow \{<\phi, \phi, \phi, \phi, \phi, \phi>\}$ " specific " " " "

Candidate Elimination Trace1 :-

Training Examples :

1. <sunny, warm, Normal, strong, warm, same>, Enjoy sport = yes

2. <sunny, warm, ~~normal~~ high, strong, warm, same>, Enjoy sport = yes.

- S_0 & G_0 are initial boundary sets.
- Training examples 1 & 2 force S boundary to become more general, as in "Find-S" Algo.
- They don't have any effect on G boundary.

S_0: {<ϕ, ϕ, ϕ, ϕ, ϕ, ϕ>}

↓

S_1: {<sunny, warm, Normal, strong, warm, same>}

↓

S_2: {<sunny, warm, ?, strong, warm, same>}

G_0, G_1, G_2: {<?, ?, ?, ?, ?, ?>}

→ Candidate_ Elimination Trace 2:-

Training Example:

3. <Rainy, Cold, High, strong, Warm, change> Enjoyspor No

∵ its a -ve example, it will now force G2 boundary to be specialised to G3. & there be several alternative maximally general hypotheses that are included.

S_2, S_3: {<sunny, warm, ?, strong, Warm, same>}

G3: {<sunny, ?, ?, ?, ?, ?>, <?, warm, ?, ?, ?, ?>, <?, ?, ?, ?, ?, same>}

G2: {<?, ?, ?, ?, ?, ?>}

→ Candidate_Elimination Trace 3:-

Training Example:

4. <Sunny, Warm, high, strong, cool, change>, Enjoy sport = yes.

∵ +ve Training example, it generalizes 'S' boundary from S3 to S4.

Also one member of G3 should be deleted since it is no longer more general than S4 boundary.

S3: {<sunny, warm, ?, strong, warm, same>}

↓

S4: {<sunny, warm, ?, strong, ?, ?>}

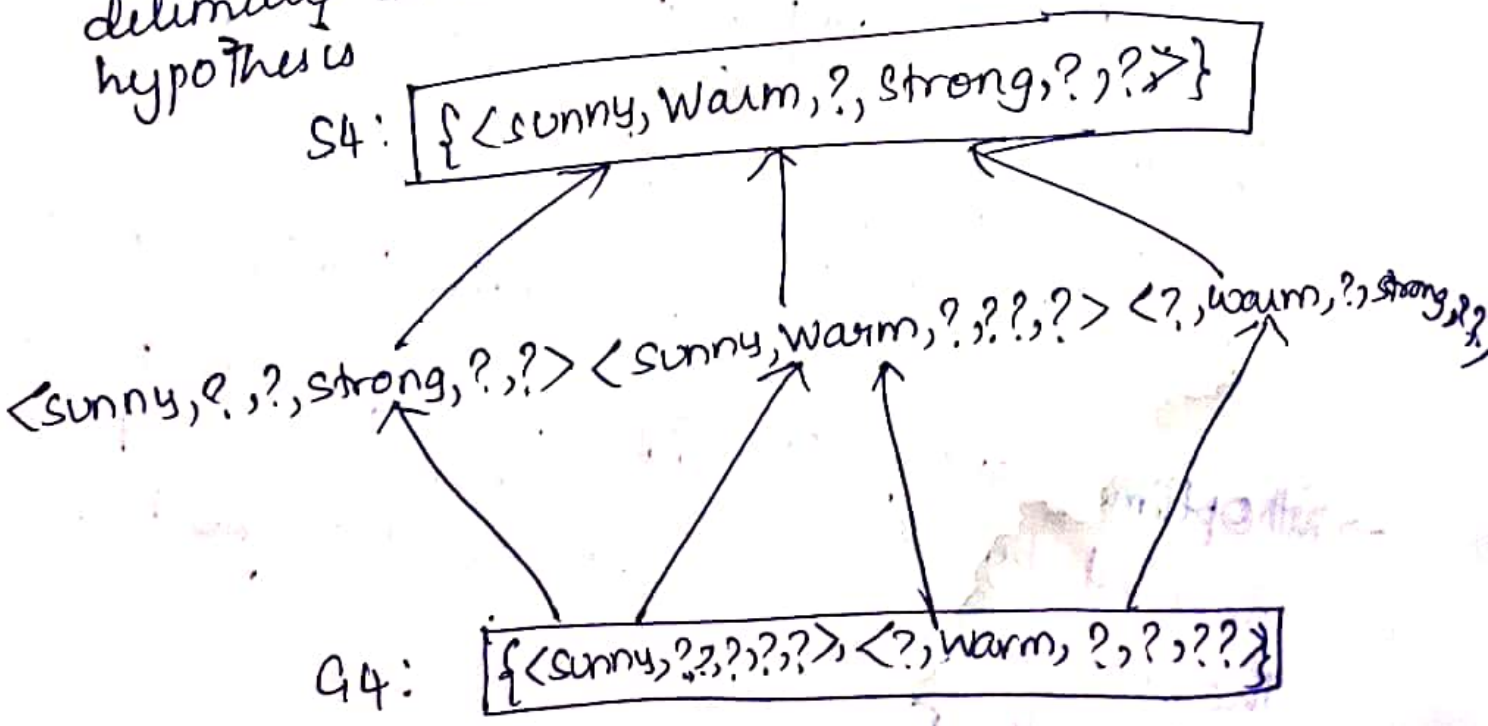
G4: {<sunny, ?, ?, ?, ?, ?> <?, warm, ?, ?, ?, ?>}

↑

G3: {<sunny, ?, ?, ?, ?, ?> <?, warm, ?, ?, ?, ?> <?, ?, ?, ?, ?, same>}

Final version :-

- Set S4 & G4 delimit version space of all hypotheses consistent with set of incrementally observed training examples.
- The foll. learned version space is independent of the seq. in which training examples present.
- S & G Boundaries move monotonically closer, delimiting smaller & smaller version space of candidate hypothesis



Remarks On Version space & "Candidate-Elimination"

→ "Version space" learned by the Candidate-Elimination algorithm will converge towards the hypothesis that correctly describes the target concept, provided
(i) There are no errors in the training examples.
(ii) There is some Hypothesis in H that correctly describes the target concept.

→ Also if training data contains errors, Algo. will remove correct target concept from Version space & may create an empty version space.

→ If the learner is allowed to conduct experiments in which it chooses next instance (a.k.a. Query) then it obtains correct classification for this instance from an external oracle (ex: Nature or Teacher)

Example:-
For "Enjoy sport" task, the "Query strategy" would be to choose an instance that would be classified +ve by some of these hypotheses, but negative by others such as:
<Sunny, Warm, Normal, Light, Warm, Same>

→ With Optimal Query strategy, size of Version space is Reduced by half with each new example. The correct target concept can therefore be found with only $\lceil \log_2 |VS| \rceil$ experiments

→ It is possible to classify certain examples with same degree of confidence as if the target concept had been uniquely identified, even if it uses partially learned concepts.

→ Consider an instance A, which was not amongst the training examples, the learner will classify it as +ve since, condition will be met iff instance satisfies every member of S. (with definition of more_general_than).

→ Illy for instance B, it will be classified Negative by every hypothesis in version space; given a partially learned concept. (Just check if instance satisfies none of members of G)

→ For instance C, Half of version space hypotheses classify it +ve & ½ as −ve. Hence Learner can't classify this instance confidently.

→ Incase of instance D, it is classified +ve by 2 of Version space hypotheses & −ve by other 4 ⇒ Negative Classification it is.

Instance	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
A	Sunny	warm	Normal	strong	cool	change	?
B	Rainy	cold	"	light	warm	same	?
C	Sunny	warm	"	"	"	"	?
D	Sunny	cold	"	strong	"	"	?

Q5. ILLUSTRATE ABOUT BASIC DECISION TREE ALGORITHM WITH A SUITABLE EXAMPLE.

(8M)

THE BASIC DECISION TREE LEARNING ALGORITHM:

- Most Algos. for learning decision Trees are built by varying the core Algo.
- The core Algo. uses Top-down, greedy search through the space of possible decision Trees.
- Few ex. are: ID3 Algo., C4.5 Algo.
- The Basic Algo. is ID3 Algo. in which it starts by questioning "which Attribute to be Tested at Root?"
- In order to Answer it, first, all the instance attributes are evaluated using a statistical test which determines how well it classifies the Training examples. Then, The best attribute is selected & used as test at "Root" node of the Tree.
- Further, a descendant to the root node will be created & training examples are sorted w.r.t it. Entire process is repeated using training examples assoc. with each descendant node to select the best attribute to test at that particular point of tree.
- Greedy Search:- The above process forms a greedy search since, algo. never back tracks to reconsider earlier choices made.

- Summary of ID3 Algo:-

ID3(Examples, Target_attribute, Attributes)
(Training examples; attribute whose value is predicted by Tree; list of other attributes)

1. Create Root Node for Tree
2. If all Examples are +ve, Return Single node tree Root, with label = +
3. " " " are -Ve, Single node tree Root, with label = -
4. If Attributes = empty ⇒ Return Single node tree Root
   label = most Common value of Target_attribute

else Begin
(i) A ← attribute from "Attributes" that best classifies "Examples"
(ii) Decision Attribute for Root ← A
(iii) for all, v_i of A
   + Add new tree branch below root corresponding to test A = v_i
   + Let Examples$_{v_i}$ be subset of Examples that have the value v_i for A.

* If $Examples_{v_i}$ is Empty
⇒ Below this new branch add a leaf node with label = most common value of "Target_attribute" in "Examples"

Else
⇒ Below this new branch add the subTree
ID3($Examples_{v_i}$, Target_attribute, Attributes − {A})

End
Return Root.

→ Which Attribute is the Best classifier?
- An Attribute that is most useful for classifying examples
- An Attribute that has "highest" information gain
- Information gain: It is a measure of how well a given attribute separates the Training examples accdg to their Target classification.

→ Entropy:
- Measures Homogeneity of Examples.
- Entropy is needed to define Info. gain.
- It characterizes purity of an arbitrary collecⁿ of examples.
- Consider 'S' = collection of +ve & -ve Examples of a target concept.

$$Entropy(S) \equiv -P_{\oplus} \log_2 P_{\oplus} - P_{\ominus} \log_2 P_{\ominus}$$

Where, $P_{\oplus}$ = proportion of +ve examples in S
& $P_{\ominus}$ = " " −ve " ".

(NOTE: $0 \log 0 = 0$)

Ex: If S has 14 ex. & 9 are +ve & 5 are −ve, then, entropy(S) $Entropy([9+, 5-]) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0.940$.

- If the members of 'S' belong to same class then Entropy = 0.

[Solu:- 'S' has 14 exs, $P_{\oplus} = 14$ & $P_{\ominus} = 0$

$\rightarrow E([14+, 0-]) = -\frac{14}{14}\log_2\frac{14}{14} - \frac{0}{14}\log_2 0$

$= -1\log_2 1 - 0 = 0$.

- If there are equal no. of +ve & -ve ex. then Entropy = 1.

- If there are Unequal no. of +ve & -ve ex. then Entropy lies b/w 0 & 1.

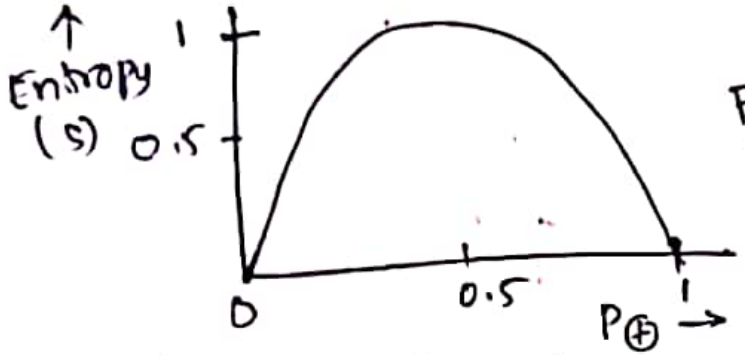


Fig:- Entropy for a relative to a boolean classification as the proportion, $P_{\oplus}$, of +ve examples varies b/w 0 & 1.

- "Entropy", accdg to info. theory, specifies min. no. of bits of info. needed in order to encode the classification of an arbitrary member of S.

  * If $P_{\oplus} = 1$ $\Rightarrow$ Receiver knows that drawn ex. will be +ve
    $\Rightarrow$ no msg need to be sent
    $\Rightarrow$ Entropy = 0.

  * If $P_{\oplus} = 0.5$ $\Rightarrow$ 1 bit is needed to indicate if drawn ex. is +ve or -ve.

  * If $P_{\oplus} = 0.8$ $\Rightarrow$ A collection of msgs. can be encoded using almost < 1 bit per msg by assigning shorter codes for +ve exs. & longer codes for less likely, -ve exs.

→ Generally, (in classification which is not boolean), the target attribute can take on 'c' diff. values, then the entropy of 'S' relative to this c-wise classification is defined as

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

where, P_i = proportion of S belonging to class i

(log is used ∵ encoded length is in bits). If target attribute can take 'c' [illegible] Entropy = $\log_2 c$

→ Info. gain Measures the expected Reduction in Entropy:-

- WKT Entropy is a measure of purity / impurity of in the collection of Training exs. (examples)
- Info. gain is a measure of effectiveness of an attribute in classifying the Training data.
- Info. gain, measures the expected reduction in entropy which is caused due to the partitioning of exs. accdg to this attribute.
- Gain(S, A) = Info. gain of an attribute A relative to a collection of examples 'S'.

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

where, Values(A) = set of all possible values for the attribute A.

S_v = subset of 's' for which attribute 'A' has the value v

$= \{ s \in S \mid A(s) = v \}$.

Entropy(S) = Entropy of original collection 'S'.

- The second term in above eqn. is Expected Entropy.

$$\text{Expected Entropy} = \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v).$$

It is the sum of entropies of each subset S_v, weighted by the fraction of examples $\frac{|S_v|}{|S|}$ that belong to S_v.

- Gain(S, A) can also be defined as info. provided about the "target func. value" given value of some other attribute A.
- It is also the no. of bits saved when encoding target value of an arbitrary member of S, by knowing the value of attribute 'A'.

Example:-

- Let S = Collection of training example described by attributes (including) like wind that can take 2 values, Weak or strong.
- Assuming, as previous, S[9+, 5−] & with that now add that, 6 +ve, 2 −ve examples that have wind = Weak & remaining all have wind = Strong
  ⇒ values (wind) = Weak, strong
-
  $S = [9+, 5-]$
  $S_{Weak} \leftarrow [6+, 2-]$
  $S_{strong} \leftarrow [3+, 3-]$

For such situation Infogain, will be,

$$Gain(S, wind) = Entropy(S) - \sum_{v \in [weak, strong]} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$= Entropy(S) - \frac{8}{14} Entropy(S_{weak}) - \frac{6}{14} Entropy(S_{strong})$$

$$= 0.940 - \frac{8}{14}\left[-\frac{6}{8}\log_2\frac{6}{8} - \frac{2}{8}\log_2\frac{2}{8}\right] - \frac{6}{14}[1] \quad [\because Entropy(S_{strong}) = 1 \text{ coz } 3+,3-]$$

$$= 0.940 - \frac{8}{14}(0.811) - \frac{6}{14}(1.)$$

$$= 0.048.$$

Illy we calculate Gain(S, Humidity) Considering [3+, 4−] as High & [6+, 1−] as Normal ⇒ Gain(S, Humidity) = 0.151

With Humidity = High

S: [9+, 5−]
E = 0.940
Humidity
High → [3+, 4−] E = 0.985
Normal → [6+, 1−] E = 0.592

Gain(S, Humidity) = 0.151

& wind

S: [9+, 5−]
E = 0.940
wind
Weak → [6+, 2−] E = 0.811
Strong → [3+, 3−] E = 1.0

Gain(S, wind) = 0.048.

Illy
Gain(S, outlook) = 0.246
& Gain(S, wind) = 0.048
Gain(S, Temperature) = 0.029

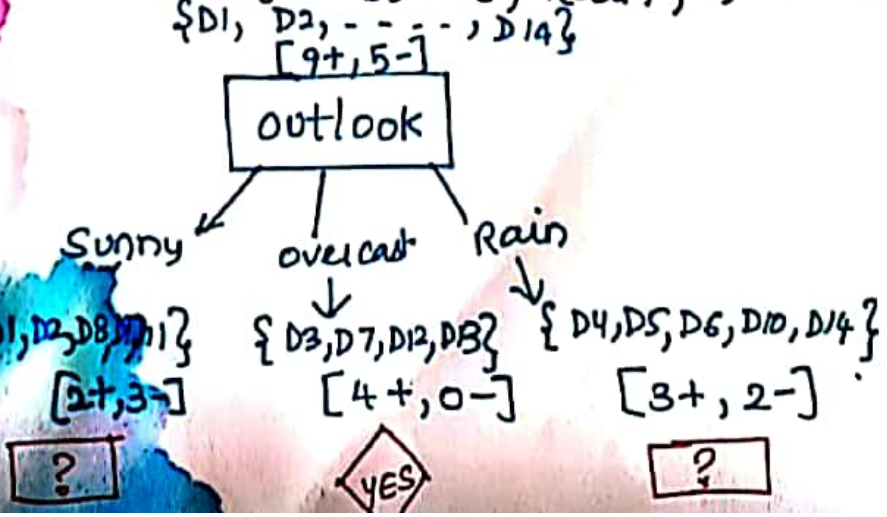
→ To understand working of ID3, let's consider following training eks.

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
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.

→ Topmost node of decision Tree is Created.
→ which attribute should be tested 1st.

→ ID3 determines Info. gain for each candidate attribute i.e outlook, Temp., Humidity & wind & selects one with Highest Info. gain i.e Here, "Outlook".

⇒ Thus "Outlook" is selected as decision attribute for root node and Branches are created for each possible value Here 3 viz. Sunny, Rain, & Overcast.



- For the Outlook = overcast, "playTennis" is always a +ve example.
  Therefore it became leaf node.
- But the process must be further repeated for new attributes at the descendant nodes of outlook = sunny & rain.
- Attributes that are included in higher part of tree are excluded. process continues for each leaf node until ① every attribute has already been included through the tree (or)
  ② Training exs. assoc. with this leaf node have all same Target attribute value (Entropy = 0)