CBCS SCHEME

USN | 1 | C | R | 1 | 9 | M | C | A | 1 | 3 |

18MCA53

Fifth Semester MCA Degree Examination, Feb./Mar. 2022
Machine Learning

Time: 3 hrs. Max. Marks: 100

Note: *Answer any FIVE full questions, choosing ONE full question from each module.*

Module-1

1 a. What do you mean by a well – posed learning problem? Explain the important features that are required to well – define a learning problem. **(10 Marks)**

b. Elaborate the design choices of choosing the training experience and choosing the target function while designing a learning system. **(10 Marks)**

OR

2 a. Illustrate find – S algorithm over Enjoy Sport Concept Training instances are given below :

Example	Sky	Air Temp	Humidity	Wind	Water	Forecast	Enjoy Sport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

(10 Marks)

b. Define Concept and Concept learning. Explain how the concept learning task determines the Hypothesis for given target concept. **(10 Marks)**

Module-2

3 a. List the advantages of Decision Tree Representation. Which problems are appropriate for Decision Tree Learning? **(10 Marks)**

b. Present the ID_3 algorithm for Decision Tree Learning. **(10 Marks)**

OR

4 a. Consider the following set of training examples :

i) What is the entropy of this collection of training examples with respect to the target function classification?

ii) What is the Information gain of a 2 relative to these training example? **(10 Marks)**

Instance	Classification	a_1	a_2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

b. Define Overfitting. How to avoid overfitting? **(10 Marks)**

Module-3

5 a. Explain in detail about the problems appropriate for neural network learning and why? **(10 Marks)**

b. Define Perceptron. Explain the concept of Single perceptron, with neat diagram. **(10 Marks)**

OR

6 a. Present the Back propagation algorithm for feed forward networks containing two layers of sigmoid units. (10 Marks)

b. Discuss the Perceptron training rule and Delta rule that solves the learning problem of perceptron. (10 Marks)

Module-4

7 a. What is Bayes theorem and Maximum posterior hypothesis? (04 Marks)

b. Derive an equation for MAP hypothesis using Bayes theorem. (06 Marks)

c. Consider a football game between two rival team : Team 0 and Team1. Suppose Team 0 wins 65% of the time and Team 1 wins the remaining matches. Among the games won by Team 0, only 30% of them come from playing on Team 1's football field. On the other hand, 75% of the victories for Team 1 are obtained while playing at home. If team 1 is to host the next match between the two teams, which team will most likely emerge as the winner? (10 Marks)

OR

8 a. Describe Brute – Force MAP learning algorithm. (06 Marks)

b. Discuss the Naive Bayes classifier. (04 Marks)

c. The following table gives data set about stolen vehicles. Using Naïve Bayes classifier classify the new data (Red , SUV , Domestic).

Color	Type	Origin	Stolen
Red	Sports	Domestic	Yes
Red	Sports	Domestic	No
Red	Sports	Domestic	Yes
Yellow	Sports	Domestic	No
Yellow	Sports	Imported	Yes
Yellow	SUV	Imported	No
Yellow	SUV	Imported	Yes
Yellow	SUV	Domestic	No
Red	SUV	Imported	No
Red	Sports	Imported	Yes

(10 Marks)

Module-5

9 a. Write short notes on the following : i) Estimating Hypothesis accuracy ii) Binomial distribution. (10 Marks)

b. Discuss the method of comparing two algorithms. Justify with Paired to – tests method. (10 Marks)

OR

10 a. Write in detail about the K – Nearest Neighbor algorithm and its approach to perform classification. (10 Marks)

b. Discuss the learning tasks and Q – learning in the context of reinforcement learning. (10 Marks)

* * * * *

1.a

Well-Posed Learning Definition: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.
Examples:
Checkers Game: A computer program that learns to play checkers might improve its performance as measured by its ability to win at the class of tasks involving playing checkers game, through experience obtained by playing games against itself:
***checkers learning problem*:**
Task T: playing checkers

Performance measure P: percent of games won against opponents

Training experience E: playing practice games against itself

A handwriting recognition learning problem:
Task T: recognizing and classifying handwritten words within images

Performance measure P: percent of words correctly classified

Training experience E: a database of handwritten words with given classifications

***A robot driving learning problem*:**
Task T: driving on public four-lane highways using vision sensors

Performance measure P: average distance travelled before an error (as judged by human overseer)

Training experience E: a sequence of images and steering commands recorded while observing a human driver

1.b.

DESIGNING A LEARNING SYSTEM
The basic design issues and approaches to machine learning are illustrated by designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament
1. Choosing the Training Experience

2. Choosing the Target Function

3. Choosing a Function Approximation Algorithm

1. Estimating training values

2. Adjusting the weights

1. Choosing the Training Experience

The first design choice is to choose the type of training experience from which the

system will learn.

The type of training experience available can have a significant impact on success or failure of the learner.

There are three attributes which impact on success or failure of the learner
1. Whether the training experience provides ***direct or indirect feedback*** regarding the choices made by the performance system.

For example, in checkers game:

In learning to play checkers, the system might learn from ***direct training examples*** consisting of ***individual checkers board states*** and ***the correct move for each***.
Indirect training examples consisting of the ***move sequences*** and ***final outcomes*** of various games played. The information about the correctness of specific moves early in the game must be inferred indirectly from the fact that the game was eventually won or lost.
Here the learner faces an additional problem of ***credit assignment***, or determining the degree to which each move in the sequence deserves credit or blame for the final outcome.
2. The degree to which the ***learner controls the sequence of training examples***

For example, in checkers game:
The learner might depends on the ***teacher*** to select informative board states and to provide the correct move for each.
Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move.
The learner may have complete control over both the board states and (indirect) training classifications, as it does when it learns by playing against itself with ***no teacher present***.
3. How well it represents the ***distribution of examples*** over which the final system performance P must be measured

For example, in checkers game:
In checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament.
If its training experience E consists only of games played against itself, there is a danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested.
It is necessary to learn from a distribution of examples that is different from those on which the final system will be evaluated.

2. Choosing the Target Function

The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program.
Let's consider a checkers-playing program that can generate the legal moves from any board state.
The program needs only to learn how to choose the best move from among these legal moves. We must learn to choose among the legal moves, the most obvious choice for the type of information to be learned is a program, or function, that chooses the best move for any given board state.

1. Let ***ChooseMove*** be the target function and the notation is
ChooseMove : B→ M
which indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.
ChooseMove is a choice for the target function in checkers example, but this function will turn out to be very difficult to learn given the kind of indirect training experience available to our system
2. An alternative target function is an ***evaluation function*** that assigns a ***numerical score*** to any given board state

Let the target function V and the notation
V:B →R
which denote that V maps any legal board state from the set B to some real value. Intend for this target function V to assign higher scores to better board states. If the system can successfully learn such a target function V, then it can easily use it to select the best move from any current board position.
Let us define the target value V(b) for an arbitrary board state b in B, as follows:
If b is a final board state that is won, then V(b) = 100

If b is a final board state that is lost, then V(b) = -100

If b is a final board state that is drawn, then V(b) = 0

If b is a not a final state in the game, then V(b) = V(b'),

Where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game

3. Choosing a Function Approximation Algorithm

In order to learn the target function **f** we require a set of training examples, each describing a specific board state b and the training value Vtrain(b) for b.
Each training example is an ordered pair of the form (b, Vtrain(b)).
For instance, the following training example describes a board state b in which black has won the game (note x2 = 0 indicates that red has no remaining pieces) and for which the target function value Vtrain(b) is therefore +100.
((x1=3, x2=0, x3=1, x4=0, x5=0, x6=0), +100)
Function Approximation Procedure
1. Derive training examples from the indirect training experience available to the learner

2. Adjusts the weights wi to best fit these training examples

1. Estimating training values

A simple approach for estimating training values for intermediate board states is to assign the training value of Vtrain(b) for any intermediate board state b to be V(Successor(b))
Where ,
V is the learner's current approximation to V
Successor(b) denotes the next board state following b for which it is again the
Rule for estimating training values
Vtrain(b) ← V (Successor(b))
2. Adjusting the weights

Specify the learning algorithm for choosing the weights wi to best fit the set of training examples {(b, Vtrain(b))}
A first step is to define what we mean by the bestfit to the training data.
One common approach is to define the best hypothesis, or set of weights, as that which minimizes the squared error E between the training values and the values predicted by the hypothesis.
Several algorithms are known for finding weights of a linear function that minimize E. One such algorithm is called the ***least mean squares, or LMS training rule***. For each observed training example it adjusts the weights a small amount in the direction that reduces the error on this training example
LMS weight update rule :- For each training example (b, Vtrain(b))
Use the current weights to calculate V (b)
For each weight wi, update it as
wi ← wi + η (Vtrain (b) - V(b)) xi

Here η is a small constant (e.g., 0.1) that moderates the size of the weight update.
Working of weight update rule
When the error (Vtrain(b)- V(b)) is zero, no weights are changed.

When (Vtrain(b) - V(b)) is positive (i.e., when V(b) is too low), then each weight

2.a

Answer: FIND-S Algorithm

1. Initialize *h* to the most specific hypothesis in *H*
2. For each positive training instance *x*

For each attribute constraint a_i in *h*

If the constraint a_i is satisfied by *x*

Then do nothing

Else replace a_i in *h* by the next more general constraint that is satisfied by *x* 3. Output hypothesis *h*

Unanswered by FIND-S

1. Has the learner converged to the correct target concept?
2. Why prefer the most specific hypothesis?
3. Are the training examples consistent?
4. What if there are several maximally specific consistent hypotheses?

1. Consider the given below following training example.

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Show the general and specific boundaries of the version space after applying candidate elimination algorithm.

Answer: CANDIDATE-ELIMINTION algorithm begins by initializing the version space to the set of all hypotheses in H;

Initializing the G boundary set to contain the most general hypothesis in H

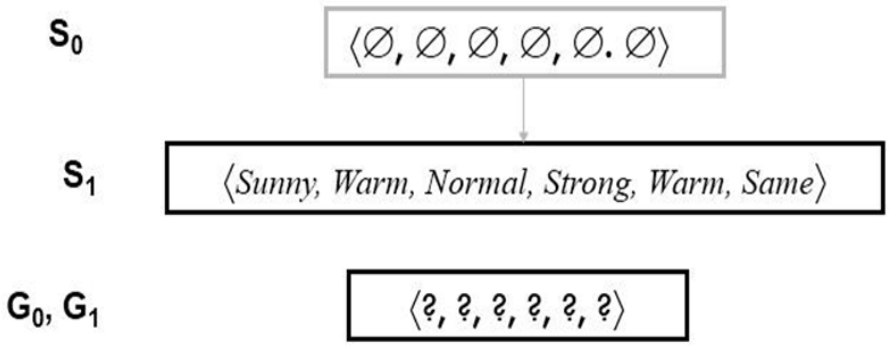
G0 (?, ?, ?, ?,?,?)

Initializing the S boundary set to contain the most specific (least general) hypothesis

S0 (Φ, Φ , Φ, Φ, Φ, Φ)

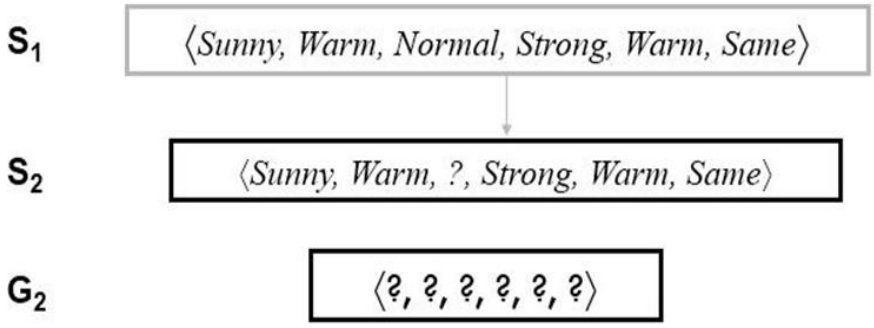
For training example d,

⟨Sunny, Warm, Normal, Strong, Warm, Same ⟩ +



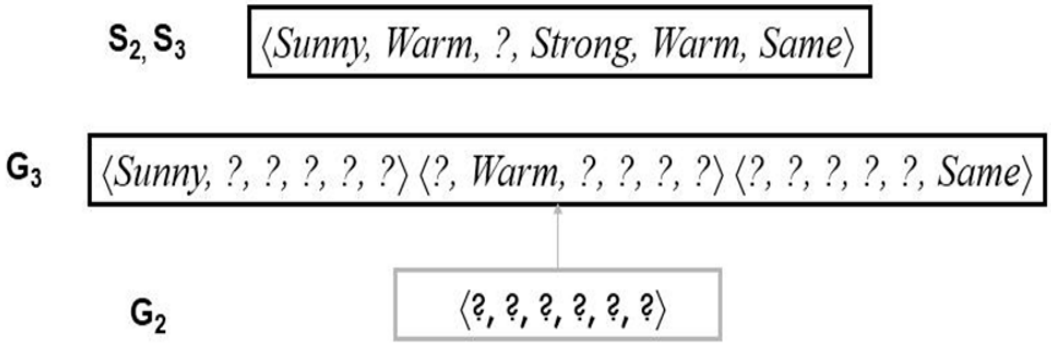
When the second training example is observed, it has a similar effect of generalizing S further to S2, leaving G again unchanged i.e., G2 = G1 = G0

For training example d,
⟨Sunny, Warm, High, Strong, Warm, Same⟩ +



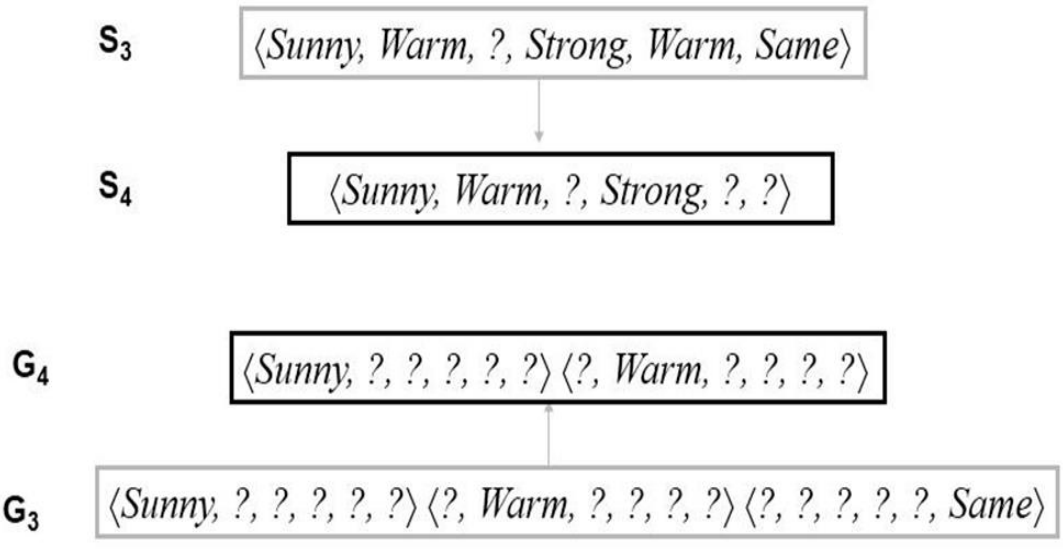
Consider the third training example

For training example d,
⟨Rainy, Cold, High, Strong, Warm, Change ⟩ –

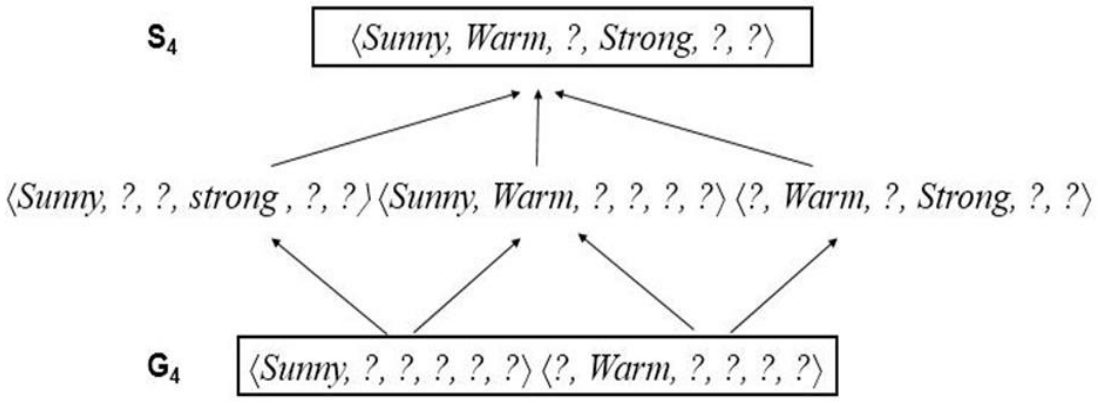


Consider the fourth training example

For training example d,
⟨Sunny, Warm, High, Strong, Cool Change ⟩ +



After processing these four examples, the boundary sets S4 and G4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples.



2.b.

What is Concept Learning…?

"A task of acquiring potential(best) hypothesis (solution) that best fits the given training examples."

Objective is to learn EnjoySport

{Sky, AirTemp, Humidity, Wind, Water, Forecast} → EnjoySport

Tom enjoys his favorite water sports

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

⇒ The set of items over which the concept is defined is called the **set of *instances*** (denoted by **X**)

⇒ The concept to be learned is called the ***Target Concept*** (denoted by c: X--> {0,1})

⇒ The set of ***Training Examples*** is a set of instances, x, along with their target concept value c(x).

⇒ Members of the concept (instances for which c(x)=1) are called ***positive examples.***

⇒ Nonmembers of the concept (instances for which c(x)=0) are called ***negative examples.***

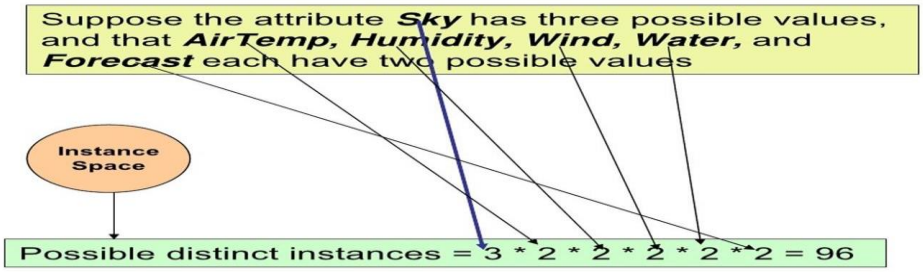
⇒ **H** represents the **set of *all possible hypotheses***. H is determined by the human designer's choice of a hypothesis representation.

⇒ **The goal of concept-learning is to find a hypothesis**

h:X --> {0,1} such that h(x)=c(x) for all x in X.

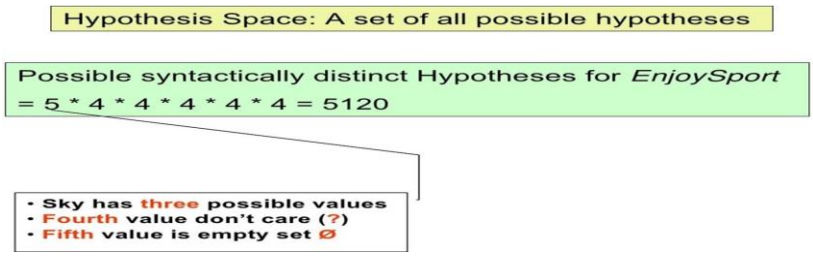
- For each attribute, the hypothesis will either
  - indicate by a **"?'** that any value is acceptable for this attribute,
  - specify a single required value (e.g., **Warm)** for the attribute, or
  - indicate by a **"ø"** that no value is acceptable.

- General hypothesis-that every day is a positive **(?, ?, ?, ?, ?, ?)**
- Specific possible hypothesis-that no day is a positive **(ø, ø, ø, ø, ø, ø)**



Instance Space:

- Consider, for example, the instances X and hypotheses H in the ***EnjoySport*** learning task.
- Given that the attribute sky has three possible values and that ***AirTemp, Humidity, Wind, Water,*** and ***Forecast*** each have two possible values, the instance space X contains exactly **3 . 2 . 2 . 2 . 2 . 2 = 96** distinct instances.



The number of semantically distinct hypothesis is only **1 + (4 . 3 . 3 . 3 . 3 . 3) =973**

3.a.

DECISION TREE REPRESENTATION

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.
- Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute.

An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node.

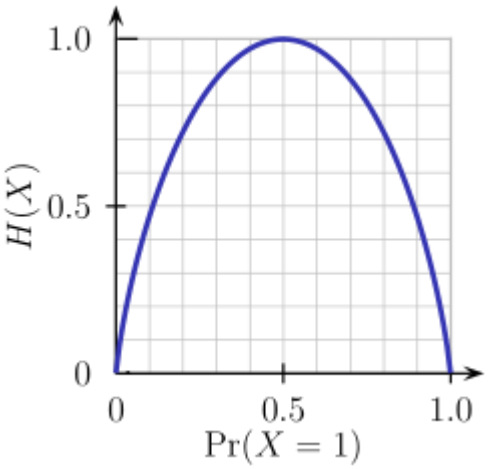
Example:-

Let's say we have a sample of 50 students with three variables Gender (Boy/ Girl), Class(X/ XI) and Height (5 to 6 ft). 20 out of these 50 play cricket in rest time. Suppose you want to find on unknown dataset which contains all the features(Gender, class, height) that he/she will play or not in rest time.

This is where decision tree supports, it will separate the students based on all values of three variable and identify the variable, which creates the best uniform sets of students.

Entropy

Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information. Flipping a coin is an example of an action that provides information that is random.



From the above graph, it is quite evident that the entropy H(X) is zero when the probability is either 0 or 1. The Entropy is maximum when the probability is 0.5 because it projects perfect randomness in the data and there is no chance if perfectly determining the outcome.

ID3 follows the rule — A branch with an entropy of zero is a leaf node and A brach with entropy more than zero needs further splitting.
Mathematically Entropy for 1 attribute is represented as:

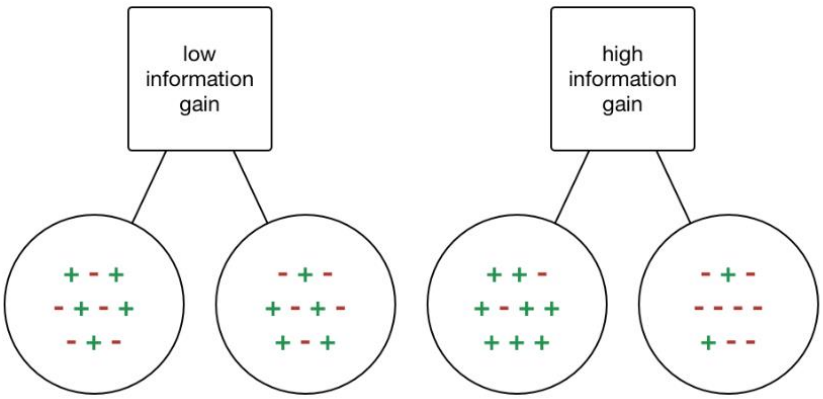
$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

Information Gain

Information gain or **IG** is a statistical property that measures how well a given attribute separates the training examples according to their target classification. Constructing a decision tree is all about finding an attribute that returns the highest information gain and the smallest entropy.



Information gain is a decrease in entropy. It computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values. ID3 (Iterative Dichotomiser) decision tree algorithm uses information gain.

Mathematically, IG is represented as:

3.b Summarize the practical issues of decision tree learning.

Answer:

Issues in learning decision trees include

1 Avoiding Overfitting the Data
2 Reduced error pruning
3 Rule post-pruning
4 Incorporating Continuous-Valued Attributes
5 Alternative Measures for Selecting Attributes
6 Handling Training Examples with Missing Attribute Values
7 Handling Attributes with Differing Costs

Avoiding Overfitting the Data

The ID3 algorithm grows each branch of the tree just deeply enough to perfectly classify the training examples but it can lead to difficulties when there is noise in the data, or when the number of training examples is too small to produce a representative sample of the true target function. This algorithm can produce trees that overfit the training examples.

How can it be possible for tree h to fit the training examples better than h', but for it to perform more poorly over subsequent examples?

Overfitting can occur when the training examples contain random errors or noise

When small numbers of examples are associated with leaf nodes.

Approaches to avoiding overfitting in decision tree learning

8 Pre-pruning (avoidance): Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
9 Post-pruning (recovery): Allow the tree to overfit the data, and then post-prune the tree

Criterion used to determine the correct final tree size

10 Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree

11 Use all the available data for training, but apply a statistical test to estimate whether expanding (or pruning) a particular node is likely to produce an improvement beyond the training set

12 Use measure of the complexity for encoding the training examples and the decision tree, halting growth of the tree when this encoding size is minimized. This approach is called the Minimum Description Length

MDL – Minimize : size(tree) + size (misclassifications(tree))

Reduced-Error Pruning

13 Reduced-error pruning, is to consider each of the decision nodes in the tree to be candidates for pruning

14 ***Pruning*** a decision node consists of removing the subtree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node

15 Nodes are removed only if the resulting pruned tree performs no worse than-the original over the validation set.

16 Reduced error pruning has the effect that any leaf node added due to coincidental regularities in the training set is likely to be pruned because these same coincidences are unlikely to occur in the validation set

2. Incorporating Continuous-Valued Attributes

Continuous-valued decision attributes can be incorporated into the learned tree.

There are two methods for Handling Continuous Attributes

17 Define new discrete valued attributes that partition the continuous attribute value into a discrete set of intervals.

E.g., {high ≡ Temp > 35º C, med ≡ 10º C < Temp ≤ 35º C, low ≡ Temp ≤ 10º C}

18 Using thresholds for splitting nodes

e.g., A ≤ a produces subsets A ≤ a and A > a

What threshold-based Boolean attribute should be defined based on Temperature?

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

19 Pick a threshold, c, that produces the greatest information gain

20 In the current example, there are two candidate thresholds, corresponding to the values of Temperature at which the value of PlayTennis changes: (48 + 60)/2, and (80 + 90)/2.

21 The information gain can then be computed for each of the candidate attributes, Temperature >54, and Temperature >85 and the best can be selected (Temperature >54)

1. Alternative Measures for Selecting Attributes

- The problem is if attributes with many values, Gain will select it ?
- Example: consider the attribute Date, which has a very large number of possible values. (e.g., March 4, 1979).
- If this attribute is added to the PlayTennis data, it would have the highest information gain of any of the attributes. This is because Date alone perfectly predicts the target attribute over the training data. Thus, it would be selected as the decision attribute for the root node of the tree and lead to a tree of depth one, which perfectly classifies the training data.
- This decision tree with root node Date is not a useful predictor because it perfectly separates the training data, but poorly predict on subsequent examples.

2. Handling Training Examples with Missing Attribute Values

The data which is available may contain missing values for some attributes Example:
Medical diagnosis

- <Fever = true, Blood-Pressure = normal, …, Blood-Test = ?, …>
- Sometimes values truly unknown, sometimes low priority (or cost too high)

Strategies for dealing with the missing attribute value

- If node n test A, assign most common value of A among other training examples sorted to node n
- Assign most common value of A among other training examples with same target value
- Assign a probability pi to each of the possible values vi of A rather than simply assigning the most common value to A(x)

5. Handling Attributes with Differing Costs

- In some learning tasks the instance attributes may have associated costs.
- For example: In learning to classify medical diseases, the patients described in terms of attributes such as Temperature, BiopsyResult, Pulse, BloodTestResults, etc.
- These attributes vary significantly in their costs, both in terms of monetary cost and cost to patient comfort

Decision trees use low-cost attributes where possible, depends only on high-cost attributes only when needed to produce reliable classifications.

4.a

Draw decision tree for the given dataset and calculate the entropy and information gain.

Instance	Classification	a1	a2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

Attribute: a1

$Values\ (a1) = T, F$

$S = [3+, 3-]$ $\quad Entropy(S) = 1.0$

$S_T = [2+, 1-]$ $\quad Entropy(S_T) = -\frac{2}{3} log_2 \frac{2}{3} - \frac{1}{3} log_2 \frac{1}{3} = 0.9183$

$S_F \leftarrow [1+, 2-]$ $\quad Entropy(S_F) = -\frac{1}{3} log_2 \frac{1}{3} - \frac{2}{3} log_2 \frac{2}{3} = 0.9183$

$Gain\ (S, a1)$ =Entropy (S) $\sum_{v \in \{T,F\}} \frac{|S_v|}{|S|} Entropy(S_v)$

$Gain(S, a1) = Entropy(S) - \frac{3}{6} Entropy(S_T) - \frac{3}{6} Entropy(S_F)$

$Gain(S, a1) = 1.0 - \frac{3}{6} * 0.9183 - \frac{3}{6} * 0.9183 = 0.0817$

Attribute: a2

$Values\ (a2) = T, F$

$S = [3+, 3-]$ $\quad Entropy(S) = 1.0$

$S_T = [2+, 2-]$ $\quad Entropy(S_T) = 1.0$

$S_F \leftarrow [1+, 1-]$ $\quad Entropy(S_F) = 1.0$

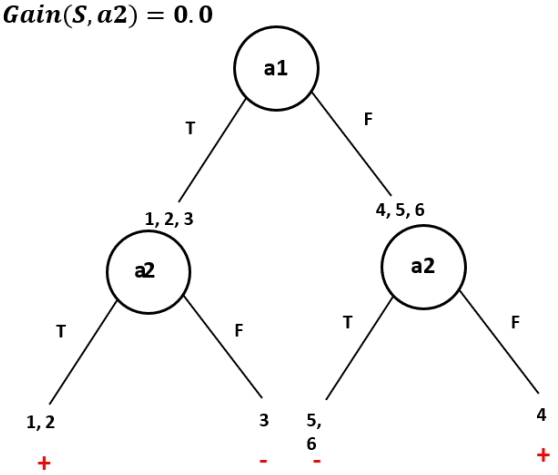
$Gain\ (S, a2) = Entropy(S) - \sum_{v \in \{T,F\}} \frac{|S_v|}{|S|} Entropy(S_v)$

$Gain(S, a2) = Entropy(S) - \frac{4}{6} Entropy(S_T) - \frac{2}{6} Entropy(S_F)$

$Gain(S, a2) = 1.0 - \frac{4}{6} * 1.0 - \frac{2}{6} * 1.0 = 0.0$

$$Gain(S, a1) = 0.0817 \quad - \quad Maximum\ Gain$$

$$Gain(S, a2) = 0.0$$



4.b.

Avoiding Overfitting the Data

The ID3 algorithm grows each branch of the tree just deeply enough to perfectly classify the training examples but it can lead to difficulties when there is noise in the data, or when the number of training examples is too small to produce a representative sample of the true target function. This algorithm can produce trees that overfit the training examples.

How can it be possible for tree h to fit the training examples better than h', but for it to perform more poorly over subsequent examples?

Overfitting can occur when the training examples contain random errors or noise

When small numbers of examples are associated with leaf nodes.

Approaches to avoiding overfitting in decision tree learning

Pre-pruning (avoidance): Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data

Post-pruning (recovery): Allow the tree to overfit the data, and then post-prune the tree

5.a

Appropriate Problems for ANN

- training data is noisy, complex sensor data
- also problems where symbolic algos are used (decision tree learning (DTL)) - ANN and DTL produce results of comparable accuracy
- instances are attribute-value pairs, attributes may be highly correlated or independent, values can be any real value
- target function may be discrete-valued, real-valued or a vector
- training examples may contain errors
- long training times are acceptable
- requires fast eval. of learned target func.
- humans do NOT need to understand the learned target func.

Instances have many attribute-value pairs: The target function to be learned is defined over instances that can be described by a vector of predefined features.

Target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes

Training examples may contain errors: ANN learning methods are quite robust to noise in the training data.

Long training times are acceptable: Network training algorithms typically require longer training times than, say, decision tree learning algorithms. Training times can range from a few seconds to many hours, depending on factors such as the number of weights in the network, the number of training examples considered, and the settings of various learning algorithm parameters.

Fast evaluation of the learned target function may be required. Although ANN learning times are relatively long, evaluating the learned network, in order to apply it to a subsequent instance, is typically very fast.

5.b

Representational Power of Perceptrons:

- The perceptron can be viewed as representing a hyperplane decision surface in the n-dimensional space of instances (i.e., points)
- The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side, as illustrated in below figure

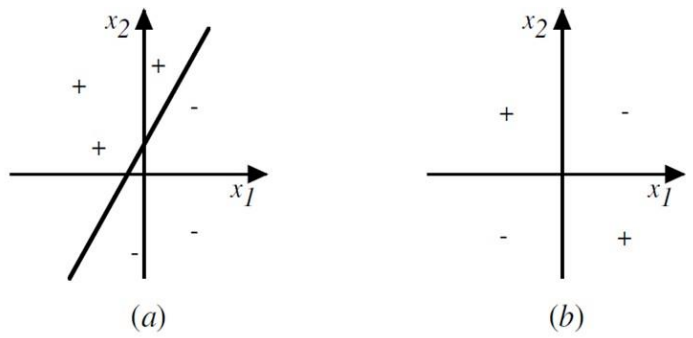


Figure : The decision surface represented by a two-input perceptron.
(a) A set of training examples and the decision surface of a perceptron that classifies them correctly. *(b)* A set of training examples that is not linearly separable.
x_1 and x_2 are the Perceptron inputs. Positive examples are indicated by "+", negative by "-".

6.a

Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
Initialize all network weights to small random numbers (e.g., between $-.05$ and $.05$).
Until the termination condition is met, Do

- For each $\langle \vec{x}, \vec{t} \rangle$ in *training_examples*, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{T4.3}$$

3. For each hidden unit h, calculate its error term δ_h

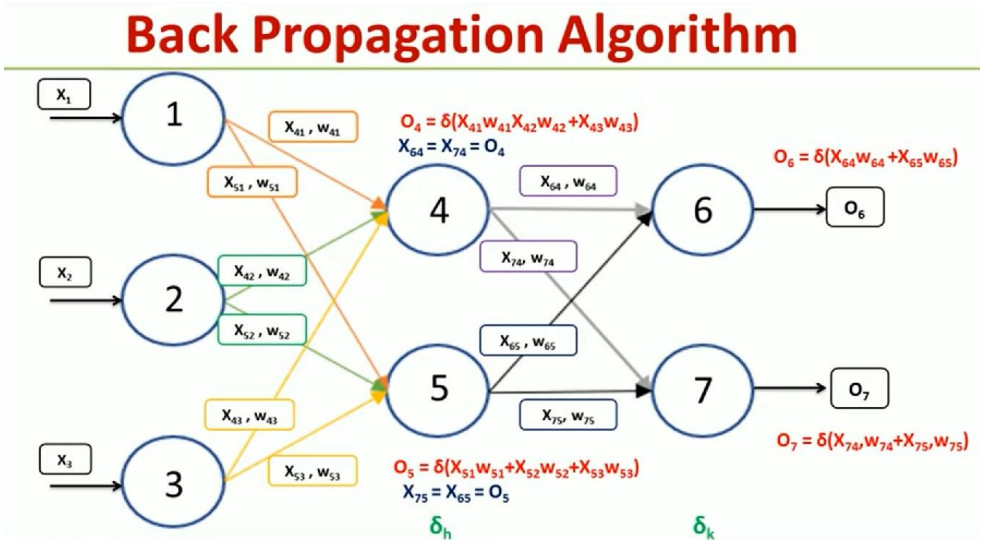
$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \tag{T4.4}$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta\, \delta_j\, x_{ji} \tag{T4.5}$$



6.b

Write the Gradient decent Algorithm and visualize the Hypothesis space for gradient decent rule.

$$\triangle w_i := \eta(t-o)x_i$$

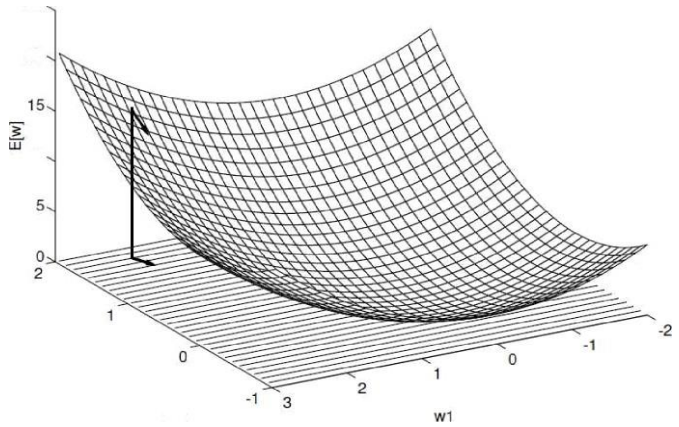
$$w_i := w_i + \triangle w_i$$

Gradient-Descent$(training\_examples, \eta)$

Each training example is a pair of the form $\langle \vec{x}, t \rangle$*, where* $\vec{x}$ *is the vector of input values, and* t *is the target output value.* η *is the learning rate (e.g., .05).*

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

- Perceptron learning converges to a consistent model if D (training set) is linearly separable.
- If the data is not linearly separable than this will not converge.
- If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept.
- The key idea behind the *delta rule* is to use *gradient descent* to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.



- Gradient descent search determines a weight vector that minimizes E by starting with an arbitrary initial weight vector, then repeatedly modifying it in small steps.
- At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface depicted in above figure. This process continues until the global minimum error is reached.

7.a

Brute-Force Bayes Concept Learning

Consider the concept learning problem

- Assume the learner considers some finite hypothesis space H defined over the instance space X, in which the task is to learn some target concept $c : X \rightarrow \{0,1\}$.
- Learner is given some sequence of training examples ((x1, d1) . . . (xm, dm)) where xi is some instance from X and where di is the target value of xi (i.e., di = c(xi)).
- The sequence of target values are written as D = (d1 . . . dm).

We can design a straightforward concept learning algorithm to output the maximum a posteriori hypothesis, based on Bayes theorem, as follows:

8.a

BRUTE-FORCE MAP LEARNING algorithm:

1. For each hypothesis h in H, calculate the posterior probability

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

2. Output the hypothesis h_{MAP} with the highest posterior probability

$$h_{MAP} = \underset{h \in H}{argmax}\ P(h|D)$$

In order specify a learning problem for the BRUTE-FORCE MAP LEARNING algorithm we must specify what values are to be used for P(h) and for P(D|h) ?

Let's choose P(h) and for P(D|h) to be consistent with the following assumptions:

- The training data D is noise free (i.e., di = c(xi))
- The target concept c is contained in the hypothesis space H

Do not have a priori reason to believe that any hypothesis is more probable than any other.

- Assumptions
  - The training data D is noise-free

$$d_i = c(x_i)$$

  - The target concept c is in the hypothesis set H

$$c \in H$$

  - All hypotheses are equally likely

$$P(h) = \frac{1}{|H|}$$

- Choice: Probability of D given h

$$P(D \mid h) = \begin{cases} 1 & \text{if } \forall d \in D, h(d) = c(d) \\ 0 & \text{else} \end{cases}$$

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$ Bayes Theorem

$$= \frac{P(D \mid h) \cdot \frac{1}{|H|}}{P(D)}$$ Given our assumptions

If the data is not consistent

$$P(h \mid D) = 0$$

If the data is consistent

$$= \frac{1 \cdot \frac{1}{|H|}}{P(D)} \quad = \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} \quad = \frac{1}{|VS_{H,D}|}$$

$VS_{H,D}$ is the version space

Given: $(\forall i \neq j)(P(h_i \wedge h_j) = 0)$ (the hypotheses are mutually exclusive):

$$P(D) = \sum_{h_i \in H} P(D \mid h_i)P(h_i)$$

$$= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|} + \sum_{h_i \notin VS_{H,D}} 0 \cdot \frac{1}{|H|}$$

$$= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|}$$

$$= \frac{|VS_{H,D}|}{|H|}$$

To summarize, Bayes theorem implies that the posterior probability P(h|D) under our assumed P(h) and P(D|h) is

$$P(D|h) = \begin{cases} \frac{1}{|V S_{H,D}|} & \text{if h is consistent with D} \\ 0 & \text{otherwise} \end{cases}$$

8.b

1. For each hypothesis h in H, calculate the posterior probability

$$P(h|D) = \frac{P(D|h)\,P(h)}{P(D)}$$

2. Output the hypothesis h_{MAP} with the highest posterior probability

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}}\, P(h|D)$$

- This algorithm may require significant computation, because it applies Bayes theorem to each hypothesis in H

– While this is impractical for large hypothesis spaces,

– The algorithm is still of interest because it provides a standard against which we may judge the performance of other concept learning algorithms.

- Brute Force MAP learning algorithm must specify values for P(h) and P(D|h).
- P(h) and P(D|h) must be chosen to be consistent with the assumptions:

1.The training data D is noise free.
2. The target concept c is contained in the hypothesis space H.
3. We have no a priori reason to believe that any hypothesis is more probable than anyother.

- Given these assumptions,
- **Given no** prior knowledge that one hypothesis is more likely than another, it is reasonable to assign the same prior probability to every hypothesis **h in H.**

$$P(h) = \frac{1}{|H|} \quad \text{for all } h \text{ in } H$$

- P(D|h) is the probability of observing the target values D = <d1 . . .dm> for the fixed set of instances <X1 . . . Xm>.

classification di given h is just 1 if di = h(xi) and 0 if di != h(xi).

- Given these choices for **P(h) and for P(D|h) we now have a fully-defined** problem for

the above **BRUTE-FORCE MAP LEARNING agorithm.**

- **Let us consider the** first step of this algorithm, which uses Bayes theorem to compute the posterior probability **P(h|D) of each hypothesis h given the observed training data D.**

- Therefore, $$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

$$P(D|h) = \begin{cases} 1 \text{ if } d_i = h(x_i) \text{ for all } d_i \text{ in } D \\ 0 \text{ otherwise} \end{cases}$$

$$P(h|D) = \frac{0 \cdot P(h)}{P(D)} = 0 \text{ if } h \text{ is inconsistent with } D$$

$$\begin{aligned} P(h|D) &= \frac{1 \cdot \frac{1}{|H|}}{P(D)} \\ &= \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} \\ &= \frac{1}{|VS_{H,D}|} \text{ if } h \text{ is consistent with } D \end{aligned}$$

Example No.	Color	Type	Origin	Stolen?
1	Red	Sports	Domestic	Yes
2	Red	Sports	Domestic	No
3	Red	Sports	Domestic	Yes
4	Yellow	Sports	Domestic	No
5	Yellow	Sports	Imported	Yes
6	Yellow	SUV	Imported	No
7	Yellow	SUV	Imported	Yes
8	Yellow	SUV	Domestic	No
9	Red	SUV	Imported	No
10	Red	Sports	Imported	Yes

NAIVE BAYES CLASSIFIER EXAMPLE

$$p(Yes) = \frac{5}{10} = 0.5$$

$$p(No) = \frac{5}{10} = 0.5$$

Color	Yes	No
Red	3/5	2/5
Yellow	2/5	3/5

Type	Yes	No
Sports	4/5	2/5
SUV	1/5	3/5

Origin	Yes	No
Domestic	2/5	3/5
Imported	3/5	2/5

$New\ Instance = (Red, SUV, Domestic)$ **No**

$$P(Yes|New\ Instance) = p(Yes) * P(Color = Red|Yes) * P(Type = SUV|Yes) * P(Origin = Domestic|Yes)$$

$$P(Yes|New\ Instance) = \frac{5}{10} * \frac{3}{5} * \frac{1}{5} * \frac{2}{5} = \frac{3}{125} = 0.024$$

$$P(No|New\ Instance) = p(No) * P(Color = Red|No) * P(Type = SUV|No) * P(Origin = Domestic|No)$$

$$P(No|New\ Instance) = \frac{5}{10} * \frac{2}{5} * \frac{3}{5} * \frac{3}{5} = \frac{9}{125} = 0.072$$

$P(No|New\ Instance) > P(Yes|New\ Instance)$

10.a

Hypothesis Testing

- Evaluates 2 mutual exclusive statement on population using sample data.
- Steps:
  1. Make initial Assumptions
  2. Collect Data
  3. Gather Evidence to Reject or accept NULL hypothesis
- What is the probability that
  error $_D$(h1) > error $_D$(h2)

COMPARING LEARNING ALGORITHMS

- Comparing the performance of 2 learning algorithms LA and LB.
- A reasonable way to define "on average" is to consider the relative performance of these 2 algorithms averaged over all the training sets of size n over Distribution D.

$$E_{S \subset D}[error_D(L_A(S)) - error_D(L_B(S))]$$

Where ,

L(S) : Hypothesis output of learning method L when given the sample S of training data .

Here S c D : The expected value is taken over samples S drawn according to the underlying instance distribution D.

Comparing learning algorithms L_A and L_B

1. Partition data D_0 into k disjoint test sets $T_1, T_2, \ldots, T_k$ of equal size, where this size is at least 30.
2. For i from 1 to k, do
   use T_i for the test set, and the remaining data for training set S_i
   - $S_i \leftarrow \{D_0 - T_i\}$
   - $h_A \leftarrow L_A(S_i)$
   - $h_B \leftarrow L_B(S_i)$
   - $\delta_i \leftarrow error_{T_i}(h_A) - error_{T_i}(h_B)$
3. Return the value $\bar{\delta}$, where

$$\bar{\delta} \equiv \frac{1}{k} \sum_{i=1}^{k} \delta_i$$

Paired t Tests

Following is the estimation procedure:

- We are given the observed values of a set of independent, identically distributed random variables Y1, Y2, ….Yk.
- We wish to estimate the mean of the probability distribution governing these Yi.
- The estimator will use sample mean which is given by

$$\bar{Y} \equiv \frac{1}{k} \sum_{i=1}^{k} Y_i$$

- We can request a new training examples drawn according to the underlying instance distribution. Modify the steps on each iteration through the loop it generates a new random training set Si and a new random test set Ti by drawing from this underlying instance distribution instead of drawing from the fixed sample D0.

Confidence interval =

$$\mu = \bar{Y} \pm t_{N,k-1}\, s_{\bar{Y}}$$

where $s_{\bar{Y}}$ is the estimated standard deviation of the sample mean

$$s_{\bar{Y}} \equiv \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (Y_i - \bar{Y})^2}$$

10.b

Discuss Q-learning algorithm with the help of an example.

Answer:

The value of Evaluation function ***Q(s, a)*** is the reward received immediately upon executing action a from state ***s***, plus the value (discounted by ***γ***) of following the optimal policy thereafter

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a)) \qquad \textbf{equ (4)}$$

Rewrite Equation (3) in terms of ***Q(s, a)*** as

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a) \qquad \textbf{equ (5)}$$

Equation (5) makes clear, it need only consider each available action ***a*** in its current state ***s*** and choose the action that maximizes ***Q(s, a)***.

An Algorithm for Learning *Q*

1. Learning the Q function corresponds to learning the **optimal policy**.

The key problem is finding a reliable way to estimate training values for ***Q***, given only a sequence of immediate rewards ***r*** spread out over time. This can be accomplished through ***iterative approximation***

$$V^*(s) = \max_{a'} Q(s, a')$$

Rewriting Equation

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a')$$

Q learning algorithm

Q learning algorithm

For each s, a initialize the table entry $\hat{Q}(s, a)$ to zero.

Observe the current state s

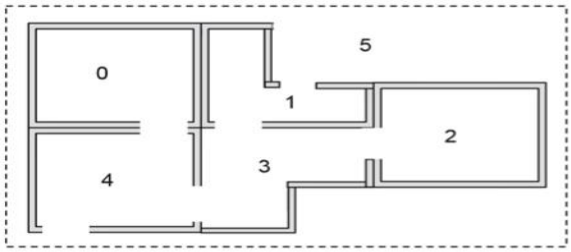
Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s, a)$ as follows:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

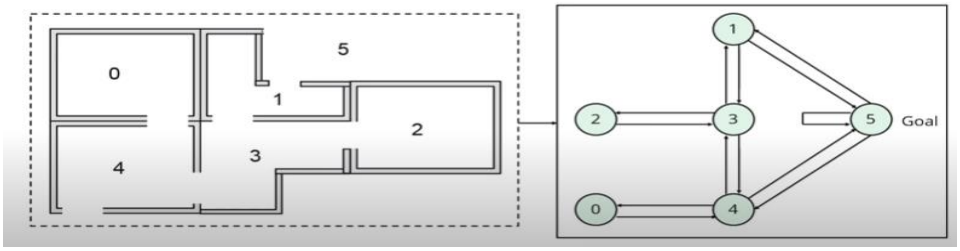
- $s \leftarrow s'$

Place an agent in any one of the rooms (0,1,2,3,4) and the goal is to reach outside the building (room 5)

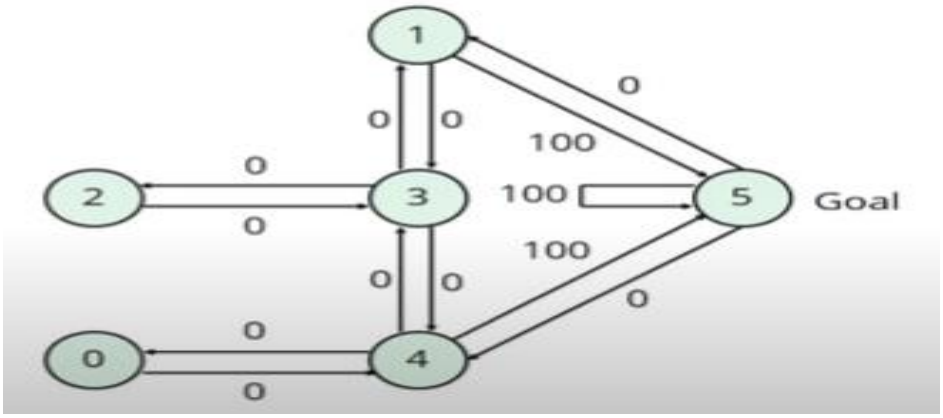


- 5 rooms in a building connected by doors
- each room is numbered 0 through 4
- The outside of the building can be thought of as one big room (5)
- Doors 1 and 4 lead into the building from room 5 (outside)

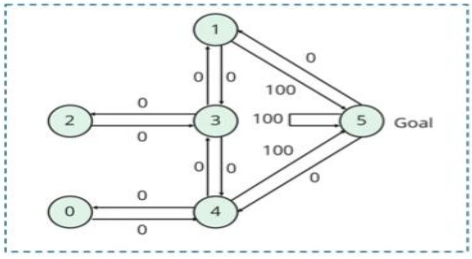
Let's represent the rooms on a graph, each room as a node, and each door as a link



Next Step is to associate reward value to each door.



We can put the state diagram and the instant reward values into a reward table, matrix R.



$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

The -1's in the table represent null values

Next Step is to create Q matrix.

Add another matrix Q, representing the memory of what the agent has learned through experience.

- The rows of matrix Q represent the current state of the agent
- columns represent the possible actions leading to the next state
- Formula to calculate the Q matrix:

*Q(state, action) = R(state, action) + Gamma * Max [Q(next state, all actions)]*

Note

The Gamma parameter has a range of 0 to 1 (0 <= Gamma > 1).

- If Gamma is closer to zero, the agent will tend to consider only immediate rewards.
- If Gamma is closer to one, the agent will consider future rewards with greater weight

Q-Learning Example:

First step is to set the value of the learning parameter Gamma = 0.8, and the initial state as Room 1.

Next, initialize matrix Q as a zero matrix:

- From room 1 you can either go to room 3 or 5, let's select room 5.
- From room 5, calculate maximum Q value for this next state based on all possible actions:

*Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]*

Q(1,5) = R(1,5) + 0.8 * Max[Q(5,1), Q(5,4), Q(5,5)] = 100 + 0.8 * 0 = 100

$Q =$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$R =$

State \ Action	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100