36. Explain planning and acting in nondeterministic domains.
37. Explain different forms of learning.
38. Explain the decision trees in learning with a neat tree diagram.
39. Describe the following about Using FOL?
i) Kinship domain ii) Numbers, sets and lists iii) The wumpus world problem

40. Formulate your opinion about inference rules for propositional logic.

Inference rules:

Inference rules are the templates for generating valid arguments. Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal.

In inference rules, the implication among all the connectives plays an important role. Following are some terminologies related to inference rules:

- **Implication:** It is one of the logical connectives which can be represented as P → Q. It is a Boolean expression.
- **Converse:** The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as Q → P.
- **Contra-positive:** The negation of converse is termed as contra-positive, and it can be represented as ¬ Q → ¬ P.
- **Inverse:** The negation of implication is called inverse. It can be represented as ¬ P → ¬ Q.

P	Q	P → Q	Q→ P	¬ Q → ¬ P	¬ P → ¬ Q.
T	T	T	T	T	T
T	F	F	T	F	T
F	T	T	F	T	F
F	F	T	T	T	T

Types of Inference rules:

1. Modus Ponens:

The Modus Ponens rule is one of the most important rules of inference, and it states that if P and P → Q is true, then we can infer that Q will be true. It can be represented as:

Notation for Modus ponens: $\frac{P \rightarrow Q, \quad P}{\therefore Q}$

Example:

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q
Statement-2: "I am sleepy" ==> P
Conclusion: "I go to bed." ==> Q.
Hence, we can say that, if P→ Q is true and P is true then Q will be true.

Proof by Truth table:

P	Q	$P \rightarrow Q$
0	**0**	**0**
0	**1**	**1**
1	**0**	**0**
1	**1**	**1**

2. Modus Tollens:

The Modus Tollens rule state that if P→ Q is true and ¬ Q is true, then ¬ P will also true. It can be represented as:

Notation for Modus Tollens: $\frac{P \rightarrow Q,\ \sim Q}{\sim P}$

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q
Statement-2: "I do not go to the bed."==> ~Q
Statement-3: Which infers that "**I am not sleepy**" => ~P

Proof by Truth table:

P	Q	$\sim P$	$\sim Q$	$P \rightarrow Q$
0	**0**	**1**	**1**	**1**
0	**1**	**1**	**0**	**1**
1	**0**	**0**	**1**	**0**
1	**1**	**0**	**0**	**1**

3. Hypothetical Syllogism:

The Hypothetical Syllogism rule state that if P→R is true whenever P→Q is true, and Q→R is true. It can be represented as the following notation:

Example:

Statement-1: If you have my home key then you can unlock my home. P→Q

Statement-2: If you can unlock my home then you can take my money. Q→R

Conclusion: If you have my home key then you can take my money. P→R

Proof by truth table:

P	Q	R	$P \rightarrow Q$	$Q \rightarrow R$	$P \rightarrow R$
0	0	0	1	1	1 ←
0	0	1	1	1	1 ←
0	1	0	1	0	1
0	1	1	1	1	1 ←
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	1 ←

4. Disjunctive Syllogism:

The Disjunctive syllogism rule state that if P∨ Q is true, and ¬P is true, then Q will be true. It can be represented as:

Notation of Disjunctive syllogism: $\frac{P \vee Q,\ \neg P}{Q}$

Example:

Statement-1: Today is Sunday or Monday. ==>P∨ Q
Statement-2: Today is not Sunday. ==> ¬P
Conclusion: Today is Monday. ==> Q

Proof by truth-table:

P	Q	$\neg P$	$P \vee Q$
0	0	1	0
0	1	1	1 ←
1	0	0	1
1	1	0	1

5. Addition:

The Addition rule is one the common inference rule, and it states that If P is true, then P∨ Q will be true.

Notation of Addition: $\frac{P}{P \vee Q}$

Example:

Statement: I have a vanilla ice-cream. ==> P

Statement-2: I have Chocolate ice-cream.

Conclusion: I have vanilla or chocolate ice-cream. ==> (P∨ Q)

Proof by Truth-Table:

P	Q	$P \vee Q$
0	0	0
1	0	1
0	1	1
1	1	1

6. Simplification:
The simplification rule state that if P∧ Q is true, then Q or P will also be true. It can be represented as:

Notation of Simplification rule: $\frac{P \wedge Q}{Q}$ Or $\frac{P \wedge Q}{P}$

Proof by Truth-Table:

P	Q	$P \wedge Q$
0	0	0
1	0	0
0	1	0
1	1	1

7. Resolution:
The Resolution rule state that if P∨ Q and ¬ P∧ R is true, then Q∨ R will also be true. It can be represented as

Notation of Resolution $\frac{P \vee Q,\ \neg P \wedge R}{Q \vee R}$

Proof by Truth-Table:

P	¬ P	Q	R	$P \vee Q$	¬ P∧R	$Q \vee R$
0	1	0	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	0	0
1	0	0	1	1	0	1
1	0	1	0	1	0	1
1	0	1	1	1	0	1

43. Explain in detail about Neural Network Architecture?

1. A neural network is composed of a number of nodes, or units, connected by links.
2. Each link has a numeric weight associated with it. Weights are the primary means of long-term storage in neural networks, and learning usually takes place by updating the weights.
3. Some of the units are connected to the external environment, and can be designated as input or output units.
4. The weights are modified so as to try to bring the network's input/output behavior more into line with that of the environment providing the inputs.
5. Each unit has a set of input links from other units, a set of output links to other units, a current activation level, and a means of computing the activation level at the next step in time,
6. given its inputs and weights. The idea is that each unit does a local computation based on inputs from its neighbors, but without the need for any global control over the set of units as a whole.
7. In practice, most neural network implementations are in software and use synchronous control to update all the units in a fixed sequence.

8. To build a neural network to perform some task, one must first decide how many units are to be used, what kind of units are appropriate, and how the units are to be connected to form a network.
9. One then initializes the weights of the network, and trains the weights using a learning
10. algorithm applied to a set of training examples for the task.
11. The use of examples also implies that one must decide how to encode the examples in terms of inputs and outputs of the network.

Network structures

- There are a variety of kinds of network structure, each of which results in very different computational properties.
- The main distinction to be made is between feed-forward and recurrent networks.
- In a feed-forward network, links are unidirectional, and there are no cycles.
- In a recurrent network, the links can form arbitrary topologies. Technically speaking, a feed-forward network is a directed acyclic graph (DAG).
- We will usually be dealing with networks that are arranged in layers.
- In a layered feed-forward network, each unit is linked only to units in the next
- layer; there are no links between units in the same layer, no links backward to a previous layer,
- and no links that skip a layer.
- The below figure shows a very simple example of a layered feed-forward network.
- This network has two layers; because the input units simply serve to pass activation to the next layer, they are not counted .

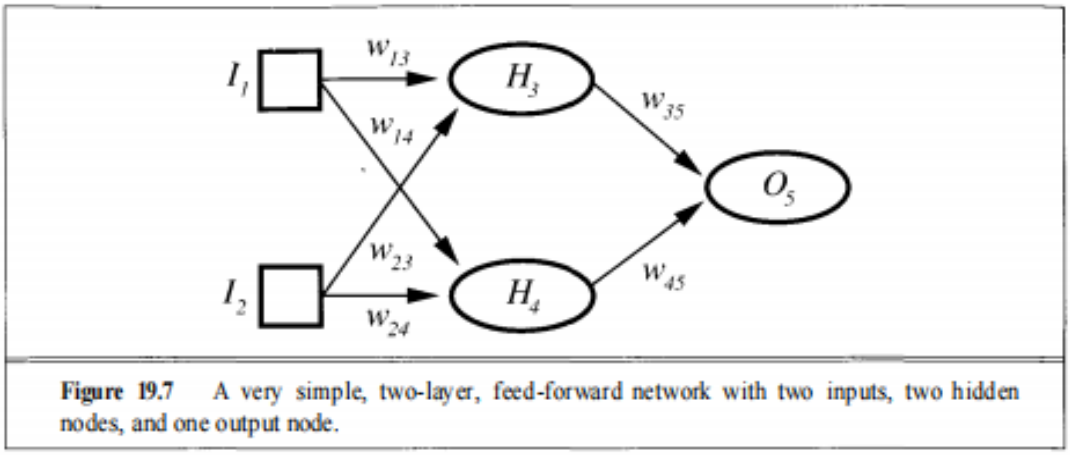


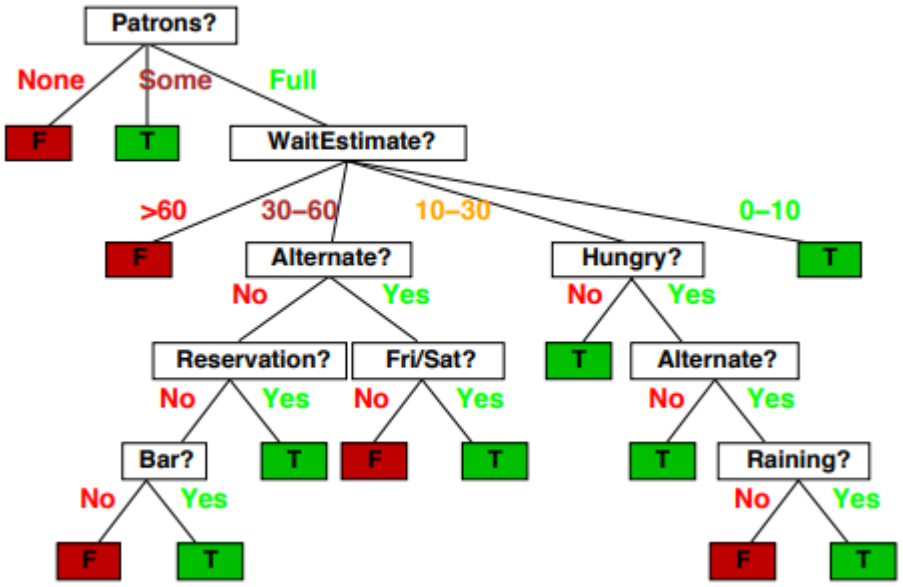
Figure 19.7 A very simple, two-layer, feed-forward network with two inputs, two hidden nodes, and one output node.

41. Explain decision tree learning with an example. What are decision rules?
42. How to use it for classifying samples?

(or)

44. Can you apply the facts to describe i) Decision tree architecture.

Decision tree induction is one of the simplest and yet most successful forms of learning algorithm.
It serves as a good introduction to the area of inductive learning, and is easy to implement.
Decision trees is one of the simplest methods for supervised learning. It can be applied to both regression & classification.
Example: A decision tree for deciding whether to wait for a place at restaurant. Target Will Wait can be True or False.



At each node of a tree, a test is applied which sends the query sample down one of the branches of the node. This continues until the query sample arrives at a terminal or leaf node. Each leaf node is associated with a value: a class label in classification, or a numeric value in regression. The value of the leaf node reached by the query sample is returned as the output of the tree.
Example:
Patrons = Full, Price = Expensive, Rain = Yes, Reservation = Yes, Hungry = Yes, Fri = No, Bar = Y es, Alternate = Yes, Type = Thai, WaitEstimate = 0 − 10
What is WillWait?
The Training Set Training set for a decision tree (each row is a sample):

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

An attribute or feature is a characteristic of the situation that we can measure. The samples could be observations of previous decisions taken.
We can re-arrange the samples as follows:

$$\underbrace{\mathbf{x}_1 = \left\{\begin{array}{c} T \\ F \\ F \\ T \\ Some \\ \$\$\$ \\ F \\ T \\ French \\ 0{-}10 \end{array}\right\},\ y_1 = T}_{sample\ 1} \quad \underbrace{\mathbf{x}_2 = \left\{\begin{array}{c} T \\ F \\ F \\ T \\ Full \\ \$ \\ F \\ F \\ Thai \\ 30{-}60 \end{array}\right\},\ y_2 = F}_{sample\ 2} \quad \cdots$$

I.e., each sample consists of a 10-dimensional vector xi of discrete feature values, and a target label yi which is Boolean.

Decision Rules

A decision rule is a simple IF-THEN statement consisting of a condition (also called antecedent) and a prediction.
For example: IF it rains today AND if it is April (condition), THEN it will rain tomorrow (prediction).
A single decision rule or a combination of several rules can be used to make predictions.

Decision rules follow a general structure:
IF the conditions are met THEN make a certain prediction.
Decision rules are probably the most interpretable prediction models.
Their IF-THEN structure semantically resembles natural language and the way we think, provided that the condition is built from intelligible features, the length of the condition is short (small number of feature=value pairs combined with an AND) and there are not too many rules.
In programming, it is very natural to write IF-THEN rules.
New in machine learning is that the decision rules are learned through an algorithm.

49. What are the basic building blocks of learning agent? Explain each of them with a neat block diagram.

(or)

45. Explain the structure of learning agent. What is the role of critic in learning

1. A learning agent can be divided into four conceptual components.
2. The most important distinction is between the learning element, which is responsible for making improvements, and the performance element, which is responsible for selecting external actions.
3. The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions.
4. The learning element takes some knowledge about the learning element and some feedback on how the agent is doing, and determines how the performance
5. element should be modified to (hopefully) do better in the future.
6. The critic is designed to tell the learning element how well the agent is doing.
7. The critic employs a fixed standard of performance. This is necessary because the percepts themselves provide no indication of the agent's success.
8. For example, a chess program may receive a percept indicating that it has checkmated its opponent, but it needs a performance standard to know that this is a good thing; the percept itself does not say so.
9. It is important that the performance standard is a fixed measure that is conceptually outside the agent; otherwise the agent could adjust its performance standards to meet its behavior.
10. The last component of the learning agent is the problem generator.

11. It is responsible for suggesting actions that will lead to new and informative experiences.
12. The point is that if the performance element had its way, it would keep doing the actions that are best, given what it
13. knows. But if the agent is willing to explore a little, and do some perhaps suboptimal actions in the short run, it might discover much better actions for the long run.
14. The design of the learning element depends very much on the design of the performance element.

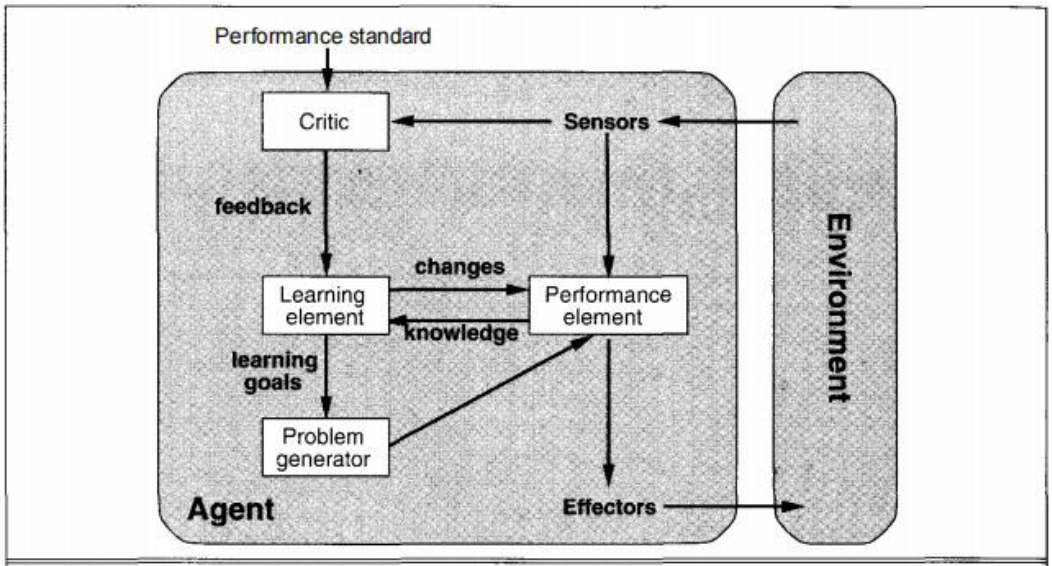
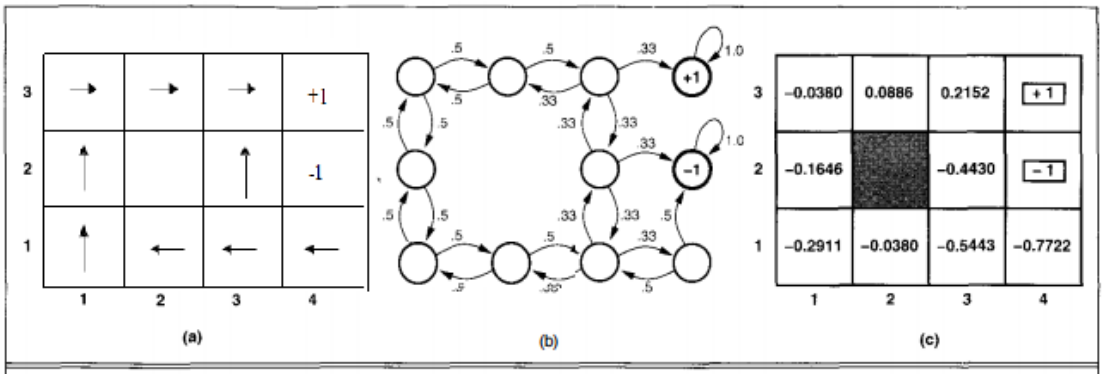


Figure 18.1 A general model of learning agents.

46. what is reinforcement learning? Explain (i) Passive reinforcement learning
(And)
47. (ii) Active reinforcement learning.
(And)
51. Explain the applications of Reinforcement learning.
(and)
52. Explain passive reinforcement learning agent with algorithm.

Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning `
and unsupervised learning.
Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

Passive reinforcement learning
1. In this learning, the agent's policy is fixed and the task is to learn the utilities of states.
2. It could also involve learning a model of the environment.
3. In passive learning, the agent's policy is fixed (i.e.) in states, it always executes the action
4. Its goal is simply to learn the utility function U (s).
5. For example: - Consider the 4 x 3 world.
6. The following figure shows the policy for that world.



(a) A simple stochastic environment. State (1,1) is the start state. (b) Each state transitions to a neighboring state with equal probability among all neighboring states. State (4,2) is terminal with reward −1, and state (4,3) is terminal with reward +1. (c) The exact utility values.

7. Clearly, the passive learning task is similar to the policy evaluation task.
8. The main difference is that the passive learning agent does not know
• Neither the transition model T (s, a,s'), which specifies the probability of reaching state's from state's after doing action a;
• Nor does it know the reward function R(s), which specifies the reward for each state.

Algorithm:

function PASSIVE-RL-AGENT(e) **returns** an action
  static: U, a table of utility estimates
    N, a table of frequencies for states
    M, a table of transition probabilities from state to state
    percepts, a percept sequence (initially empty)

  add e to *percepts*
  increment N[STATE[e]]
  $U \leftarrow$ UPDATE(U, e, *percepts*, M, N)
  if TERMINAL?[e] **then** *percepts* $\leftarrow$ the empty sequence
  return the action *Observe*

Active reinforcement learning

1. In the case of active reinforcement learning, it is the agent's decision to perform a certain task as there is no fixed policy that can perform.
2. A passive learning agent has a fixed policy that determines its behaviour.
3. "An active agent must decide what actions to do"
4. An ADP agent can be taken an considered how it must be modified to handle this new freedom.
5. The following are the required modifications:
• First the agent will need to learn a complete model with outcome probabilities for all actions. The simple learning mechanism used by PASSIVE-ADP-AGENT will do just fine for this.
• Next, take into account the fact that the agent has a choice of actions. The utilities it needs to learn are those defined by the optimal policy.

$$U(s) = R(s) + \gamma \max_{a} \sum_{s'} T(s,a,s')U(s')$$

These equations can be solved to obtain the utility function U using he value iteration or policy iteration algorithms.
• Having obtained a utility function U that is optimal for the learned model, the agent can extract an optimal action by one-step look ahead to maximize the expected utility;
• Alternatively, if it uses policy iteration, the optimal policy is already available, so it should simply execute the action the optimal policy recommends.

Algorithm:

```
function ACTIVE-ADP-AGENT(e) returns an action
  static: U, a table of utility estimates
          M, a table of transition probabilities from state to state for each action
          R, a table of rewards for states
          percepts, a percept sequence (initially empty)
          last-action, the action just executed

  add e to percepts
  R[STATE[e]] — REWARD[e]
  M — UPDATE-ACTIVE-MODEL(M, percepts, last-action)
  U — VALUE-ITERATION(U, M, R)
  if TERMINAL?[e] then
     percepts — the empty sequence
  last-action ← PERFORMANCE-ELEMENT(e)
  return last-action
```

Applications of Reinforcement Learning

Here are applications of Reinforcement Learning:

- Robotics for industrial automation.
- Business strategy planning
- Machine learning and data processing
- It helps you to create training systems that provide custom instruction and materials according to the requirement of students.
- Aircraft control and robot motion control

50. What is Non-parametric machine learning(NML)?Explain any one NML algorithm.

1. A **nonparametric model** is one that cannot be characterized by a bounded set of parameters. For example, suppose that each hypothesis we generate simply retains within itself all of the training examples and uses all of them to predict the next example.
2. Non-parametric Models are statistical models that do not often conform to a normal distribution, as they rely upon continuous data, rather than discrete values.
3. Non-parametric statistics often deal with ordinal numbers, or data that does not have a value as fixed as a discrete number.
4. The term non-parametric does not mean that the value lack inherent parameters, but rather that the parameters are flexible and can vary.
5. One may turn to non-parametric statistics when dealing with ranked data, in which some of the value of the variables is the order in which they are organized.
6. Such a hypothesis family would be nonparametric because the effective number of parameters is unbounded it grows with the number of examples.
7. This approach is called **instance-based learning** or **memory-based learning**. The simplest instance-based learning method is **table lookup**: take all the training examples, put them in a lookup table, and then when asked for h(**x**), see if **x** is in the table; if it is, return the corresponding y. The problem with this method is that it does not generalize well: when **x** is not in the table all it can do is return some default value.

Different NML Algos are:
Nearest neighbor models

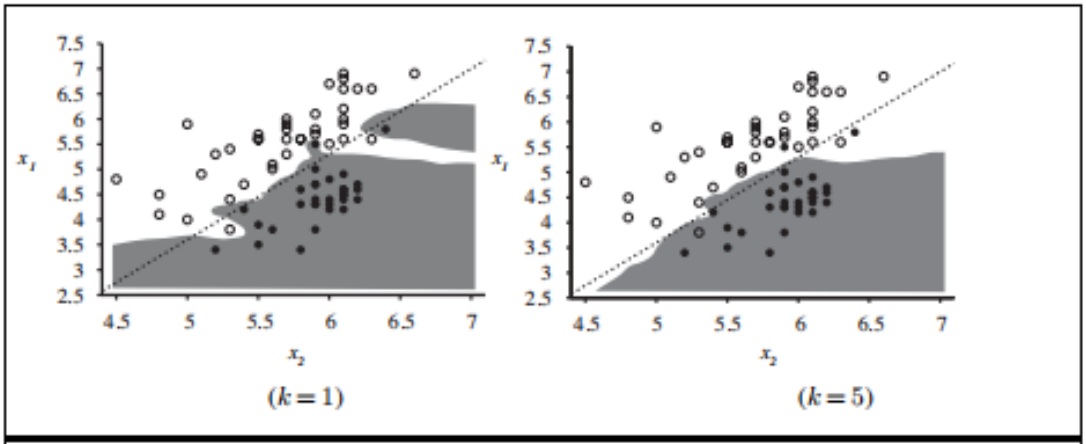


Figure 18.26 (a) A k-nearest-neighbor model showing the extent of the explosion class for the data in Figure 18.15, with $k = 1$. Overfitting is apparent. (b) With $k = 5$, the overfitting problem goes away for this data set.

We can improve on table lookup with a slight variation: given a query **x**q, fifind the k examples that are *nearest* to **x**q. This is called k-**nearest neighbors** lookup. We'll use the notation NN (k, **x**q) to denote the set of k nearest neighbors.

To do classifification, fifirst fifind NN (k, **x**q), then take the plurality vote of the neighbors. (which is the majority vote in the case of binary classifification). To avoid ties, k is always chosen to be an odd number. To do regression, we can take the mean or median of the k neighbors, or we can solve a linear regression problem on the neighbors.

In Figure , we show the decision boundary of k-nearest-neighbors classification for k = 1 and 5 on the earthquake data set.

Finding nearest neighbors with k-d trees

A balanced binary tree over data with an arbitrary number of dimensions is called a k-d tree, for k-dimensional tree.
The construction of a k-d tree is similar to the construction of a one-dimensional balanced binary tree.
We start with a set of examples and at the root node we split them along the ith dimension by testing whether xi ≤ m.
We chose the value m to be the median of the examples along the ith dimension; thus half the examples will be in the left branch of the tree and half in the right.
We then recursively make a tree for the left and right sets of examples,
stopping when there are fewer than two examples left.
To choose a dimension to split on at each node of the tree, one can simply select dimension i mod n at level i of the tree.

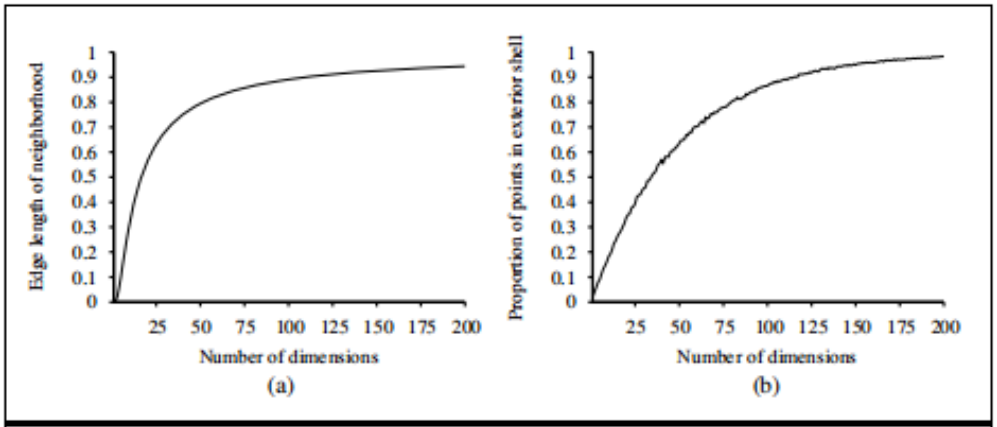


Figure 18.27 The curse of dimensionality: (a) The length of the average neighborhood for 10-nearest-neighbors in a unit hypercube with 1,000,000 points, as a function of the number of dimensions. (b) The proportion of points that fall within a thin shell consisting of the outer 1% of the hypercube, as a function of the number of dimensions. Sampled from 10,000 randomly distributed points.

Locality-sensitive hashing

Hash tables have the potential to provide even faster lookup than binary trees.
Hash codes randomly distribute values among the bins, but we want to have near points grouped together in the same bin; we want a locality-sensitive hash (LSH).
We can't use hashes to solve NN (k, xq) exactly, but with a clever use of randomized algorithms, we can find an approximate solution.
First we define the approximate nearneighbors problem: given a data set of example points and a query point xq, find, with high
probability, an example point (or points) that is near xq.
To be more precise, we require that
if there is a point xj that is within a radius r of xq, then with high probability the algorithm
will find a point xj` that is within distance c r of q.
If there is no point within radius r then the algorithm is allowed to report failure.
The values of c and "high probability" are parameters of the algorithm.

Non-parametric regression

In (a), we have perhaps the simplest method of all, known informally as "connect-the-dots," and superciliously as "piecewiselinear nonparametric regression."
This model creates a function h(x) that, when given a
query xq, solves the ordinary linear regression problem with just two points: the training examples immediately to the left and right of xq.
When noise is low, this trivial method is actually not too bad, which is why it is a standard feature of charting software in spreadsheets.

But when the data are noisy, the resulting function is spiky, and does not generalize well. k-nearest-neighbors regression (Figure 18.28(b)) improves on connect-the-dots.
Instead of using just the two examples to the left and right of a query point xq, we use the k nearest neighbors (here 3).
A larger value of k tends to smooth out the magnitude of the spikes, although the resulting function has discontinuities.
In (b), we have the k-nearestneighbors average: h(x) is the mean value of the k points, yj/k.
Notice that at the outlying points, near x = 0 and x = 14, the estimates are poor because all the evidence comes from one side (the interior), and ignores the trend.
In (c), we have k-nearest-neighbor linear regression, which finds the best line through the k examples.
This does a better job of capturing trends at the outliers, but is still discontinuous.
In both (b) and (c), we're left with the question of how to choose a good value for k. The answer, as usual, is cross-validation.

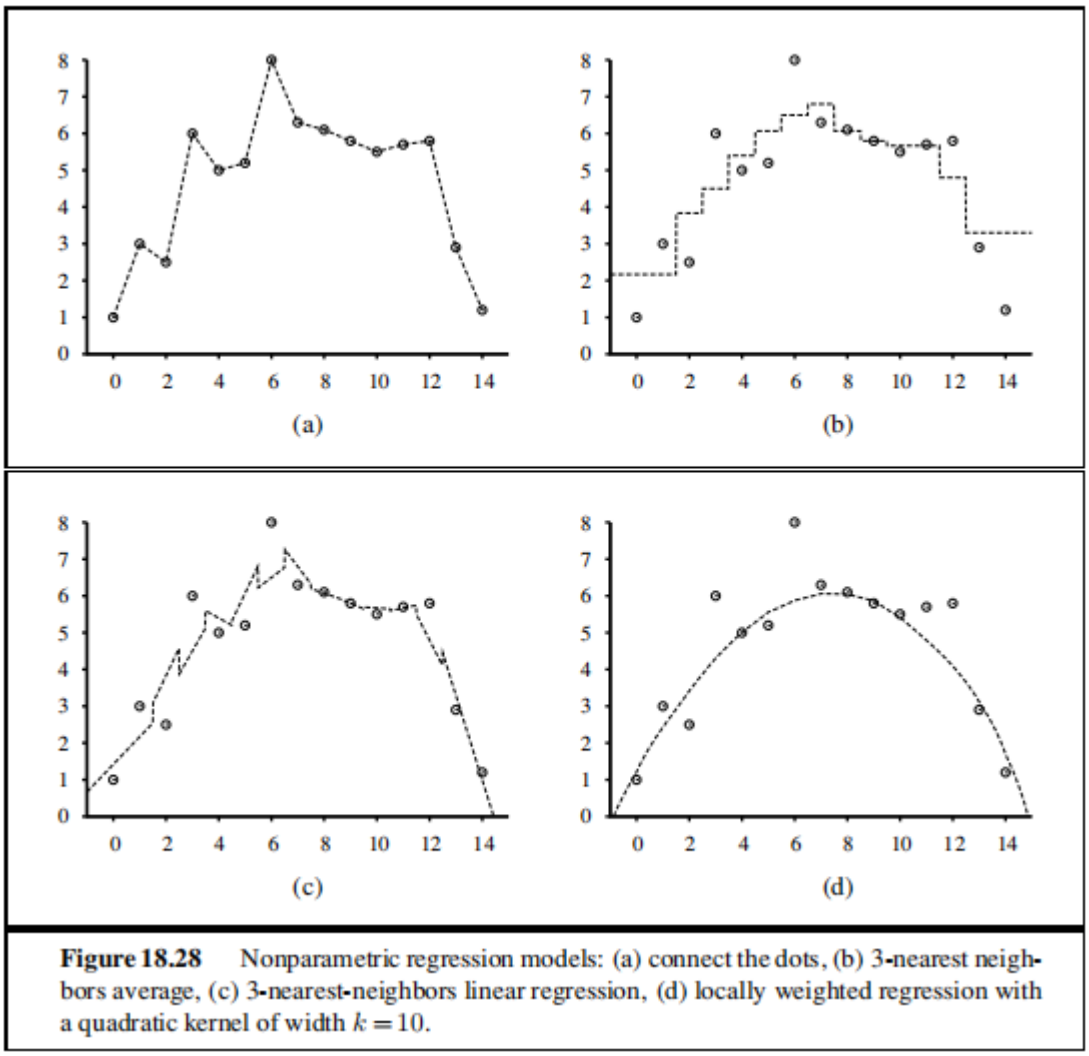


Figure 18.28 Nonparametric regression models: (a) connect the dots, (b) 3-nearest neighbors average, (c) 3-nearest-neighbors linear regression, (d) locally weighted regression with a quadratic kernel of width $k = 10$.

48. Explain the learning with hidden variables: the EM algorithm.

(or)

53. Explain the learning with hidden variables: the EM algorithm.

Expectation-Maximization algorithm can be used for variables that are not directly observable and are actually inferred from the values of the other observed variables(**latent or hidden variables**) in order to predict their values with the condition that the general form of probability distribution governing those latent variables is known to us.

This algorithm is actually at the base of many unsupervised clustering algorithms in the field of machine learning.

For example, medical records often include the observed symptoms, the physician's diagnosis, the treatment applied, and perhaps the outcome of the
treatment, but they seldom contain a direct observation of the disease itself!.

One might ask, "If the disease is not observed, why not construct a model without it?" The answer appears in below figure , which shows a small, fictitious diagnostic model for heart disease.

There are three observable predisposing factors and three observable symptoms.

Assume that each variable has three possible values (e.g., none, moderate, and severe).

Removing the hidden variable from the network in (a) yields the network in (b); the total number of parameters increases from 78 to 708.

Thus, latent variables can dramatically reduce the number of parameters required to specify a Bayesian network.

This, in turn, can dramatically reduce the amount of data needed to learn the parameters.

Hidden variables are important, but they do complicate the learning problem.

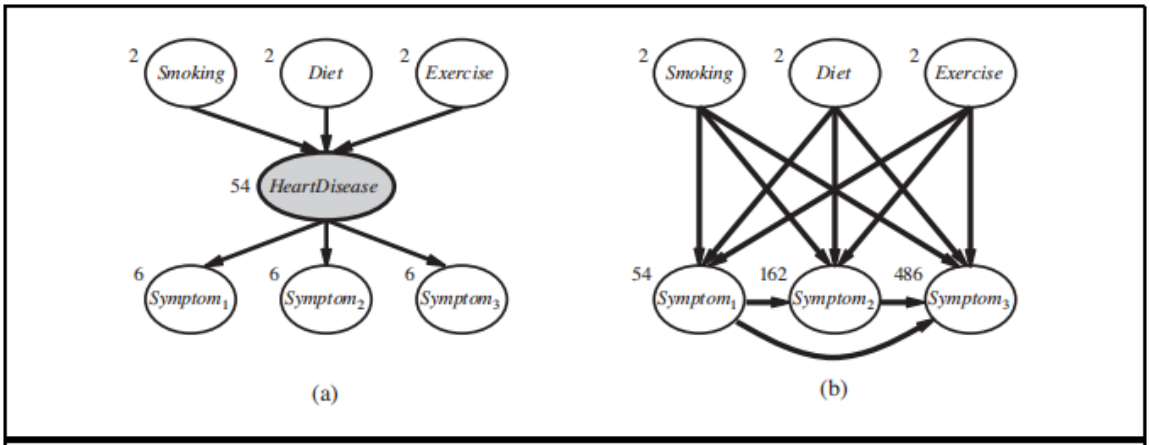


Figure 20.10 (a) A simple diagnostic network for heart disease, which is assumed to be a hidden variable. Each variable has three possible values and is labeled with the number of independent parameters in its conditional distribution; the total number is 78. (b) The equivalent network with *HeartDisease* removed. Note that the symptom variables are no longer conditionally independent given their parents. This network requires 708 parameters.

Algorithm:

1. Given a set of incomplete data, consider a set of starting parameters.
2. Expectation step (E – step): Using the observed available data of the data set, estimate (guess) the values of the missing data.
3. Maximization step (M – step): Complete data generated after the expectation (E) step is used in order to update the parameters.
4. Repeat step 2 and step 3 until convergence.

Flow chart: