

AI & ML Algorithm (315330)

Model Question Paper — Set C · Model Answers

Q.1

a) Fields where AI is applied.

2 M

- **Healthcare** — diagnosis, medical imaging, robotic surgery.
- **Transportation** — self-driving cars, traffic prediction.
- **Education** — intelligent tutoring, automatic evaluation.
- **Agriculture** — crop disease detection, yield prediction.
- **Security** — face recognition, cyber-threat detection; also banking, gaming, e-commerce.

(Any four, $\frac{1}{2}$ M each)

b) Heuristic function.

2 M

A **heuristic function $h(n)$** gives an **estimate of the cost** of the cheapest path from node n to the goal. It uses problem-specific knowledge to guide informed search (greedy best-first, A^*) towards the goal faster. $h(\text{goal}) = 0$.

Example: In route finding, $h(n)$ = straight-line distance from city n to the destination; in the 8-puzzle, $h(n)$ = number of misplaced tiles.

(Definition 1 M, example 1 M)

c) Forward vs backward chaining.

2 M

Forward chaining	Backward chaining
Starts from known facts and moves towards the goal.	Starts from the goal and moves backward to the facts.
Data-driven, bottom-up; breadth-first.	Goal-driven, top-down; depth-first.
Used in planning / control systems.	Used in diagnosis, PROLOG, MYCIN.

(Any two points, 1 M each)

d) Stages of ML life cycle.

2 M

1) Gathering data → 2) Data preparation → 3) Data wrangling → 4) Analyse data → 5) Train the model → 6) Test the model → 7) Deployment.

(Correct stages in order 2 M)

e) Overfitting and underfitting.

2 M

Overfitting: the model learns the training data along with its noise, so it performs very well on training data but poorly on new test data (high variance).

Underfitting: the model is too simple to capture the pattern, so it performs poorly on both training and test data (high bias).

(Each definition 1 M)

f) F1-score and accuracy.

2 M

F1-score is the harmonic mean of precision and recall: $F1 = 2 \times \text{Precision} \times \text{Recall} / (\text{Precision} + \text{Recall})$; it balances both and is useful for imbalanced classes.

Accuracy is the fraction of all predictions that are correct: $\text{Accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{TN} + \text{FP} + \text{FN})$.

(Each definition with formula 1 M)

g) MAE and RMSE.

2 M

MAE (Mean Absolute Error) = $(1/n) \sum |y_i - \hat{y}_i|$ — the average of absolute differences between actual and predicted values.

RMSE (Root Mean Squared Error) = $\sqrt{[(1/n) \sum (y_i - \hat{y}_i)^2]}$ — square root of the mean of squared errors; gives more weight to large errors.

(Each definition with formula 1 M)

Q.2

a) Two heuristic search techniques; greedy best-first trace.

4 M

1) Generate-and-test: generate a possible solution (a state or a path), test whether it is a goal; if yes stop, otherwise generate the next candidate. With a heuristic, the generator proposes the most promising candidates first. Simple, but slow for large spaces (like exhaustive search).

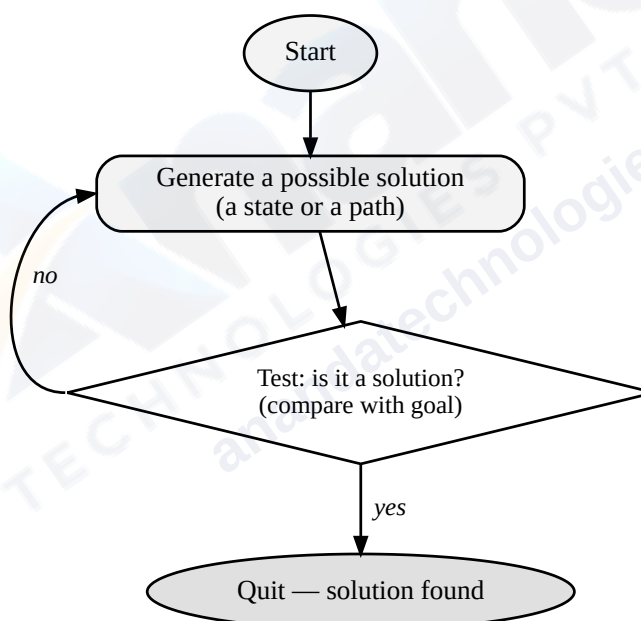


Fig.: Generate-and-test algorithm

2) Greedy best-first search: expands the node that appears closest to the goal, using $f(n) = h(n)$ only (priority queue on h). Fast, but not optimal.

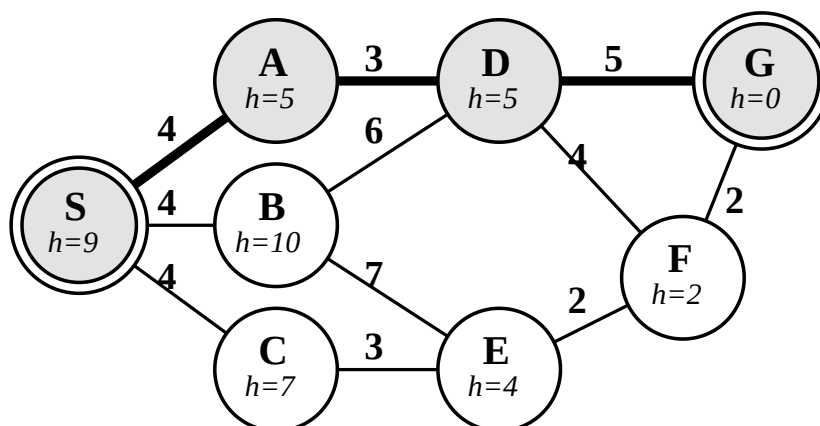


Fig.: Given graph; bold = path chosen by greedy best-first search

Step	Node selected (removed from Open)	Open list node(h), sorted on h	Closed list
1	S (h=9)	A(5), C(7), B(10)	S
2	A (h=5)	D(5), C(7), B(10)	S, A
3	D (h=5)	G(0), F(2), C(7), B(10)	S, A, D
4	G (h=0)	Goal reached — stop	S, A, D, G

Path: S → A → D → G, cost = 12. (Greedy ignores g(n); see Q.3 (a): A* finds a cheaper path.)

(Two techniques 2 M, trace 2 M)

b) Naive Bayes classification — numerical.

4 M

Naive Bayes: $P(C|X) \propto P(C) \times \prod P(x_i|C)$ — attributes are assumed independent given the class.

Prior: $P(\text{Yes}) = 6/10$, $P(\text{No}) = 4/10$

Conditional probability	Count	Value
$P(\text{Age} = \text{Youth} \text{Yes})$	2/6	0.3333
$P(\text{Income} = \text{High} \text{Yes})$	4/6	0.6667
$P(\text{Student} = \text{No} \text{Yes})$	2/6	0.3333
$P(\text{Age} = \text{Youth} \text{No})$	3/4	0.7500
$P(\text{Income} = \text{High} \text{No})$	1/4	0.2500
$P(\text{Student} = \text{No} \text{No})$	4/4	1.0000

$$P(X|\text{Yes}) \cdot P(\text{Yes}) = 6/10 \times 2/6 \times 4/6 \times 2/6 = \mathbf{0.0444}$$

$$P(X|\text{No}) \cdot P(\text{No}) = 4/10 \times 3/4 \times 1/4 \times 4/4 = \mathbf{0.0750}$$

Normalising: $P(\text{Yes}|X) = 0.0444 / 0.1194 = 0.3721$; $P(\text{No}|X) = 0.6279$

Since $P(\text{No}|X)$ is larger, the customer **will not buy** a computer (Buys = No).

(Priors 1 M, likelihoods 1 M, posteriors 1 M, decision 1 M)

c) Statistics vs data mining.

4 M

Statistics	Data mining
Science of collecting, organising, analysing and interpreting numerical data.	Process of discovering hidden patterns and knowledge from large datasets.

Starts with a hypothesis and tests it (confirmatory).	Explores data to find patterns without a prior hypothesis (exploratory).
Works well on small samples, mostly numeric data.	Works on very large databases; numeric, text, images etc.
Based on mathematical theory — probability, distributions, sampling.	Uses ML, AI, database and statistical techniques together.
Little or no data cleaning stage; data is planned and collected for the study.	Data cleaning / preprocessing is a major step (data comes from many sources).
Output: estimates, tests, confidence intervals.	Output: rules, clusters, classifications, associations.

(Any four points, 1 M each)

d) Logistic regression.

4 M

Logistic regression is a supervised **classification** algorithm (despite its name) used to predict the probability that an input belongs to a class (binary: 0/1, yes/no).

- Compute a linear combination: $z = b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_n$.
- Pass z through the **sigmoid (logistic) function** $\sigma(z) = 1 / (1 + e^{-z})$, which maps any value to a probability between 0 and 1 (S-shaped curve).
- Decision: if $\sigma(z) \geq 0.5 \rightarrow$ class 1, otherwise class 0 (threshold can be changed).
- Coefficients are learned by maximising likelihood / minimising **log-loss (cross-entropy)** using gradient descent. For more than two classes: multinomial (softmax), One-vs-Rest or One-vs-One.
- Example: predict whether an e-mail is spam, whether a student will pass, whether a tumour is malignant.

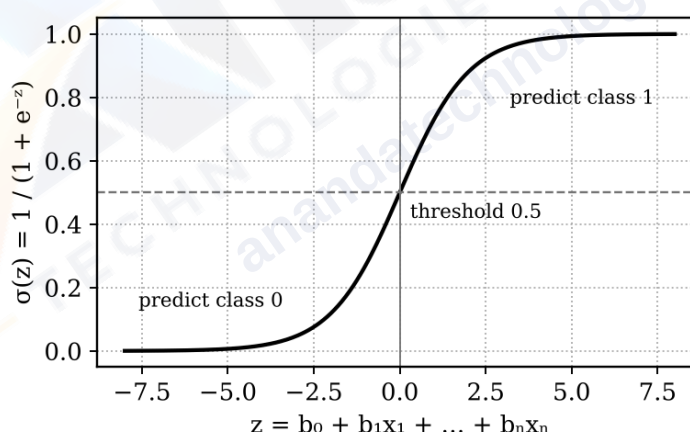


Fig.: Sigmoid function used in logistic regression

(Explanation 2 M, sigmoid with graph 1 M, example 1 M)

Q.3

a) Properties of A*; optimal path.

4 M

- **Evaluation function:** $f(n) = g(n) + h(n)$.
- **Complete:** A* always finds a solution if one exists (finite branching factor, positive step costs).

- **Optimal:** A* returns the least-cost path if $h(n)$ is **admissible** (never over-estimates the true cost: $h(n) \leq h^*(n)$); for graph search h should be **consistent** ($h(n) \leq c(n, n') + h(n')$).
- **Optimally efficient:** no other optimal algorithm using the same heuristic expands fewer nodes.
- **Complexity:** time and space are exponential, $O(b^d)$, in the worst case; it keeps all generated nodes in memory (main drawback).

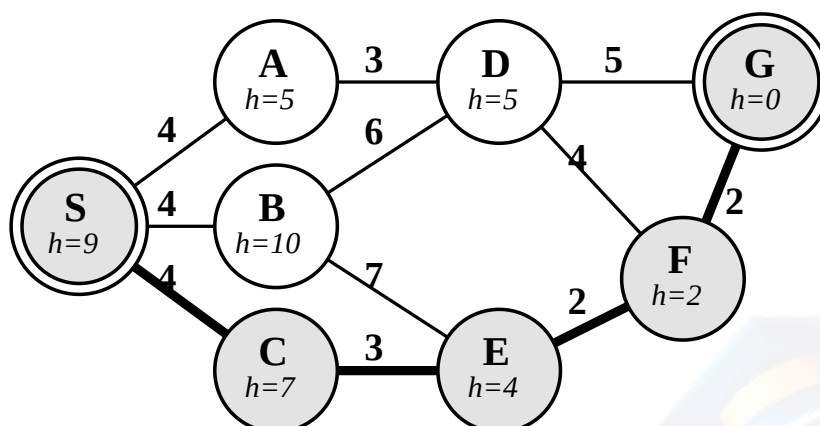


Fig.: Graph of Q.2 (a); bold = optimal path found by A*

Step	Node selected (removed from Open)	Open list node($f = g + h$), sorted on f	Closed list
1	S ($g=0, h=9, f=9$)	A(9), C(11), B(14)	S
2	A ($g=4, h=5, f=9$)	C(11), D(12), B(14)	S, A
3	C ($g=4, h=7, f=11$)	E(11), D(12), B(14)	S, A, C
4	E ($g=7, h=4, f=11$)	F(11), D(12), B(14)	S, A, C, E
5	F ($g=9, h=2, f=11$)	G(11), D(12), B(14)	S, A, C, E, F
6	G ($g=11, h=0, f=11$)	Goal reached — stop	S, A, C, E, F, G

A* path: $S \rightarrow C \rightarrow E \rightarrow F \rightarrow G$, cost = **11**, which is cheaper than the greedy path (12) — A* is optimal because the given h is admissible.

(Properties 2 M, trace and path 2 M)

b) Knowledge-based agent — architecture and need.

4 M

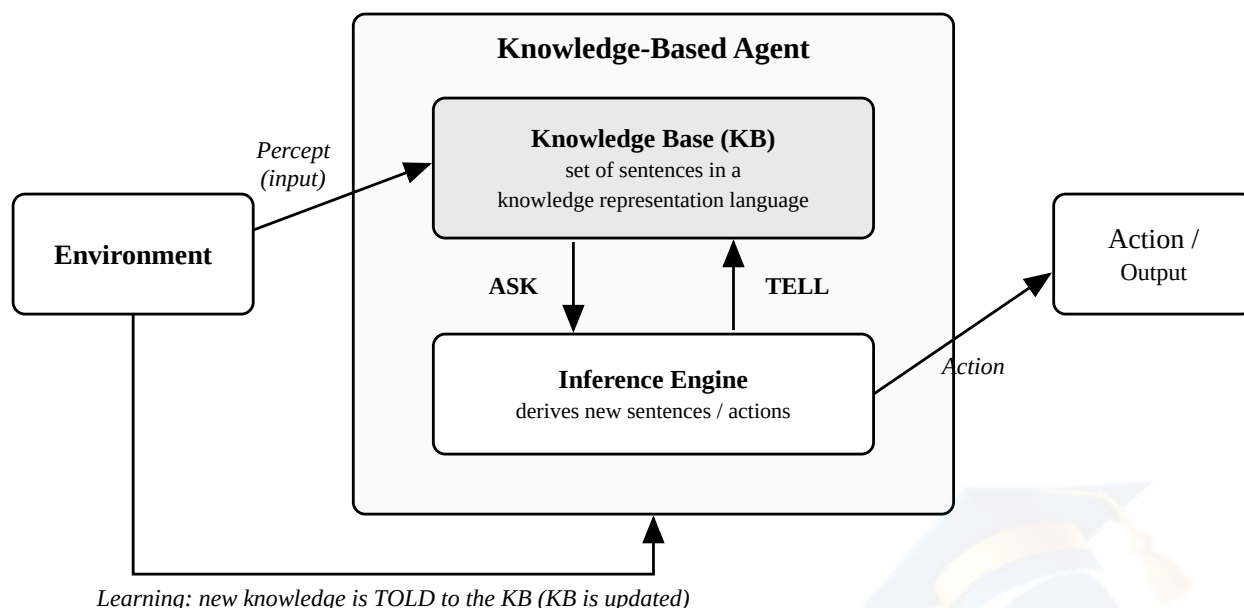


Fig.: Architecture of a knowledge-based agent

A knowledge-based agent has two main parts: the **knowledge base (KB)** — a set of sentences (facts and rules) in a formal language, and the **inference engine** — which derives new sentences and decides actions. It works through **TELL** (add percepts / new knowledge to KB) and **ASK** (query KB for the best action). Each time step: TELL(percept) → ASK(action) → TELL(action done).

Need: it can represent knowledge about the world, reason to derive hidden facts, accept new tasks as explicit goals, learn new knowledge (update KB) and adapt to changes in the environment — e.g. expert systems, Wumpus-world agent.

(Diagram 2 M, explanation 1 M, need 1 M)

c) AI vs ML.

4 M

Artificial Intelligence	Machine Learning
Technology to create machines that simulate human intelligence and behaviour.	Subset of AI that lets machines learn automatically from past data.
Aim: build smart systems that solve complex problems like humans.	Aim: learn from data to make accurate predictions on new data.
Scope is very wide — reasoning, planning, NLP, vision, robotics, ML.	Scope is limited to learning from data.
Can work with rule-based logic, search and knowledge without learning.	Always needs (large) data to learn; no data, no learning.
Types: weak (narrow) AI, general AI, super AI.	Types: supervised, unsupervised, reinforcement learning.
Examples: Sophia robot, chess engine, expert systems.	Examples: Google search ranking, auto-tagging photos, spam filter.

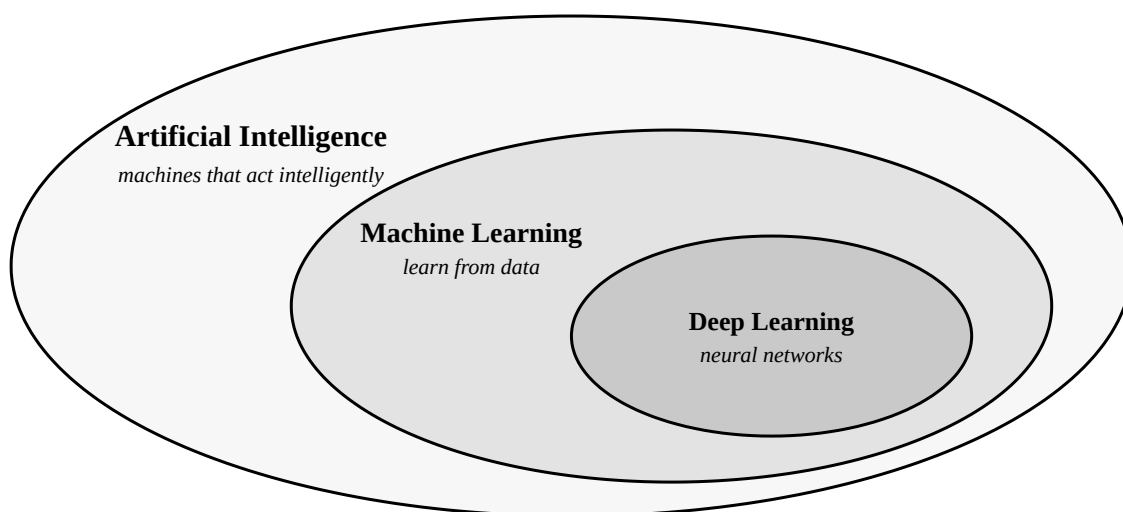


Fig.: AI $\supset$ Machine Learning $\supset$ Deep Learning

(Any four points, 1 M each)

d) K-means clustering in Python.

4 M

K-means: choose k, initialise k centroids, assign each point to the nearest centroid, recompute centroids as the mean of their points, and repeat until assignments do not change.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

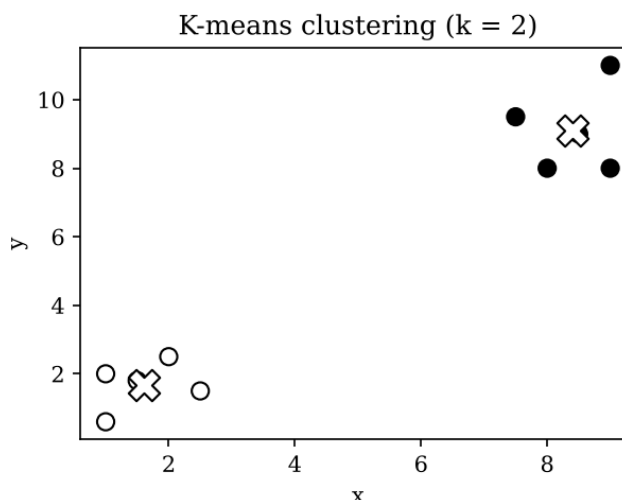
X = np.array([[1, 2], [1.5, 1.8], [2, 2.5], [8, 8], [8.5, 9], [9, 8],
              [1, 0.6], [9, 11], [2.5, 1.5], [7.5, 9.5]])
km = KMeans(n_clusters=2, n_init=10, random_state=0)
km.fit(X)

print("Cluster labels :", km.labels_)
print("Cluster centres:")
print(km.cluster_centers_.round(3))
print("Inertia (WCSS) :", round(km.inertia_, 3))
print("New point [2, 3] belongs to cluster", km.predict([[2, 3]])[0])

plt.figure(figsize=(4.5, 3.2))
plt.scatter(X[:, 0], X[:, 1], c=km.labels_, cmap='gray', edgecolors='k', s=60)
plt.scatter(km.cluster_centers_[0], km.cluster_centers_[1], marker='x',
            s=200, c='white', edgecolors='k')
plt.title("K-means clustering (k = 2)"); plt.xlabel("x"); plt.ylabel("y")
plt.show()
```

Output:

```
Cluster labels : [1 1 1 0 0 0 1 0 1 0]
Cluster centres:
[[8.4  9.1 ]
 [1.6  1.68]]
Inertia (WCSS) : 11.588
New point [2, 3] belongs to cluster 1
```



Output window: clusters and centres (X)

(Program 3 M, output 1 M)

Q.4

a) Types of agents.

4 M

- **Simple reflex agent:** selects action only on the **current percept** using condition–action rules (IF dirty THEN suck); ignores history; works only in fully observable environments.
- **Model-based reflex agent:** keeps an **internal state** (model of how the world evolves and what its actions do) to handle partially observable environments.
- **Goal-based agent:** has **goal** information and chooses actions that achieve the goal, using search and planning.
- **Utility-based agent:** uses a **utility function** to choose the action giving the best (happiest / most efficient) outcome when there are many ways to reach the goal.
- **Learning agent:** learns from experience — has learning element, performance element, critic and problem generator.

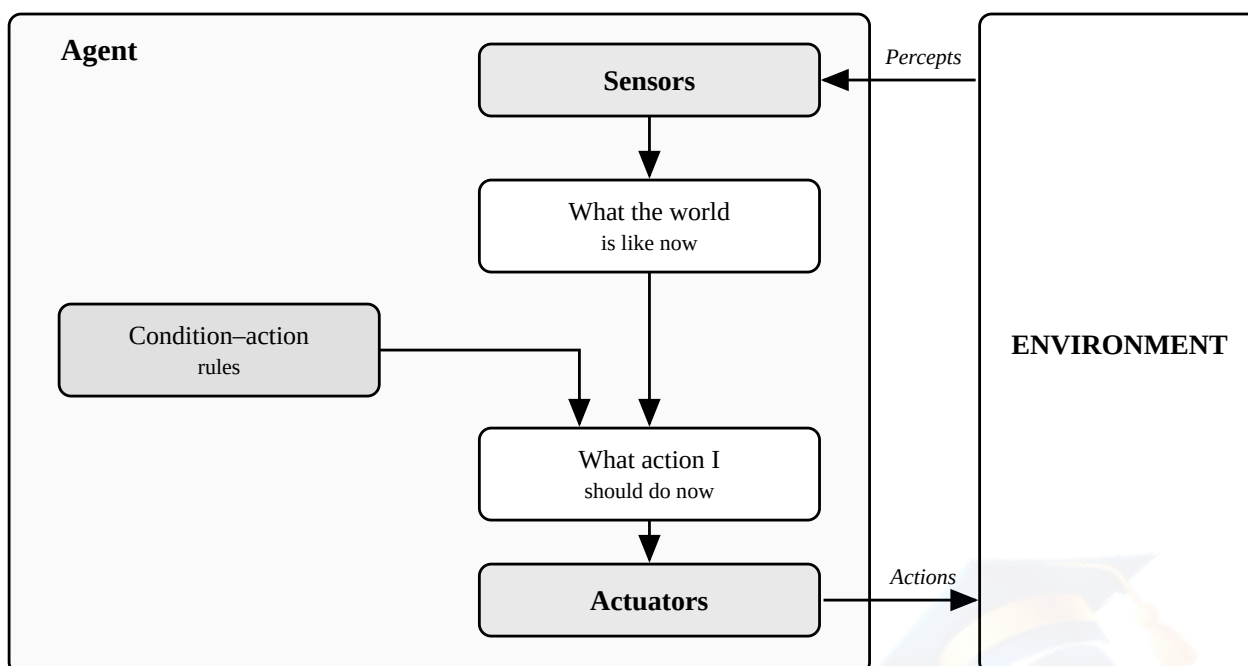


Fig.: Simple reflex agent

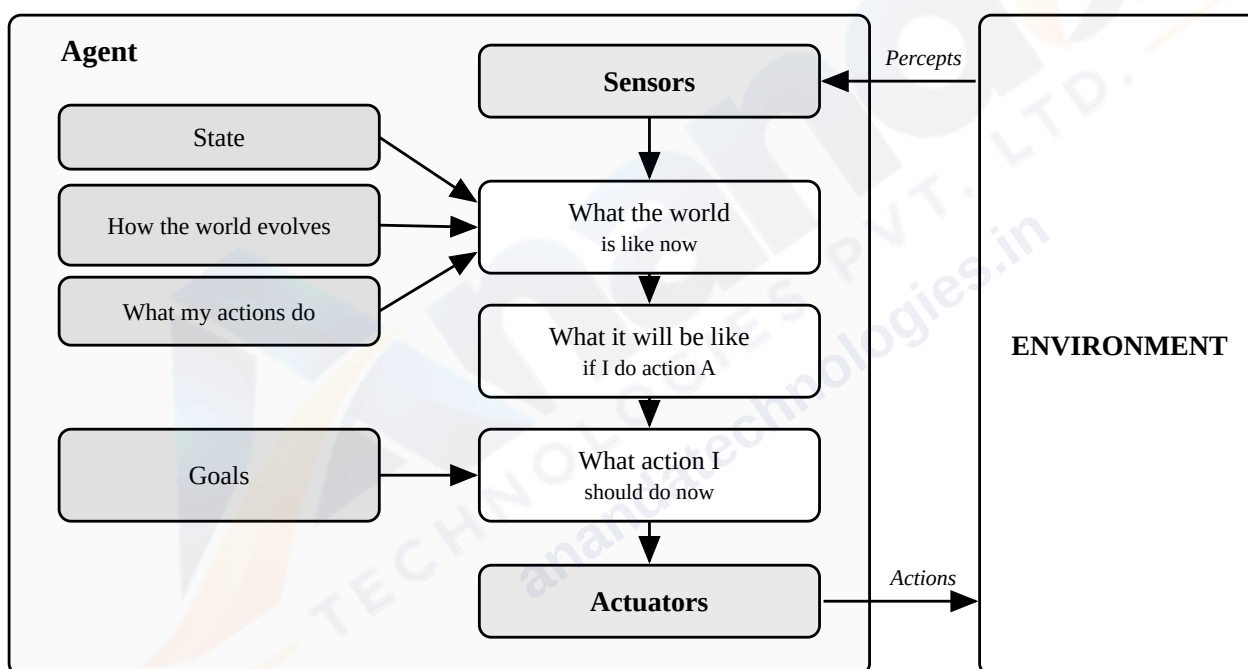


Fig.: Goal-based agent

(Types 2 M, diagrams 2 M)

b) Uncertainty; causes; need of probabilistic reasoning.

4 M

Uncertainty is the situation in which the agent cannot be sure whether a fact or rule is true — knowledge is incomplete, imprecise or unreliable, so logic with only true/false values is not enough (e.g. 'if fever then flu' is not always true).

Causes of uncertainty:

- Information from unreliable sources; noisy or faulty sensors (equipment fault).
- Incomplete / partial observation of the environment.

- Experimental errors and variations in conditions; climate change, random events.
- Laziness and theoretical ignorance — too many rules / exceptions to list.

Need for probabilistic reasoning: it combines probability theory with logic to represent degrees of belief (0 to 1), to reason when outcomes are unpredictable, when the specification of predicates is too large, and when unknown errors occur; beliefs are updated with Bayes' rule as new evidence arrives (Bayesian networks).

(Uncertainty 1 M, causes 2 M, need 1 M)

c) Training vs testing dataset.

4 M

Basis	Training dataset	Testing dataset
Purpose	To fit / build the model (learn parameters).	To evaluate the trained model.
Size	Large portion ($\approx 70-80\%$).	Small portion ($\approx 20-30\%$).
When used	During the learning phase.	After training is complete.
Labels	Labels are used by the algorithm to learn.	Labels used only to compare with predictions.
Result	Training accuracy / loss (may be optimistic).	Test accuracy — true measure of generalisation.

(Any four points, 1 M each)

d) Benefits of cross-validation; Python program.

4 M

Benefits:

- More reliable estimate of performance on unseen data than one train/test split.
- Uses all data for both training and testing — good for small datasets.
- Reduces the effect of a lucky / unlucky split (lower variance of estimate).
- Helps detect overfitting and select the best model / hyper-parameters (e.g. k in k-NN).

```
from sklearn.datasets import load_iris
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.neighbors import KNeighborsClassifier

X, y = load_iris(return_X_y=True)
model = KNeighborsClassifier(n_neighbors=5)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=1) # 5 folds
scores = cross_val_score(model, X, y, cv=cv, scoring='accuracy')

for i, s in enumerate(scores, 1):
    print(f"Fold {i}: accuracy = {s:.4f}")
print("Mean accuracy =", round(scores.mean(), 4), " Std =", round(scores.std(), 4))
```

Output:

```
Fold 1: accuracy = 0.9667
Fold 2: accuracy = 1.0000
Fold 3: accuracy = 0.9333
Fold 4: accuracy = 1.0000
```

Fold 5: accuracy = 0.9333
Mean accuracy = 0.9667 Std = 0.0298

(Benefits 2 M, program with output 2 M)

e) Entropy and information gain — numerical.

4 M

Approved: Yes = 5, No = 3 (total 8).

$$E(S) = -(5/8)\log_2(5/8) - (3/8)\log_2(3/8) = 0.9544$$

Attribute	Value	Yes	No	Entropy $E(S_v)$	Weighted E = $\sum(S_v / S)E(S_v)$	Gain = $E(S) - \text{weighted E}$
Income	High	2	0	0 (pure node)	0.6887	0.2657
	Medium	2	1	$-(2/3)\log_2(2/3) - (1/3)\log_2(1/3) = 0.9183$		
	Low	1	2	$-(1/3)\log_2(1/3) - (2/3)\log_2(2/3) = 0.9183$		
Credit	Good	4	0	0 (pure node)	0.4056	0.5488
	Bad	1	3	$-(1/4)\log_2(1/4) - (3/4)\log_2(3/4) = 0.8113$		

Gain(Income) = 0.2657, Gain(Credit) = 0.5488. **Credit** has the higher information gain, so **Credit** is selected as the root node.

(E(S) 1 M, entropy of values 1 M, gains 1 M, root 1 M)

Q.5

a) Artificial neural network; neuron output.

6 M

An **artificial neural network (ANN)** is a computing model inspired by the biological brain. It consists of many simple processing units called **neurons (nodes)** arranged in layers and connected by weighted links.

weights w on every connection; each neuron: $y = f(\sum w_i x_i + b)$

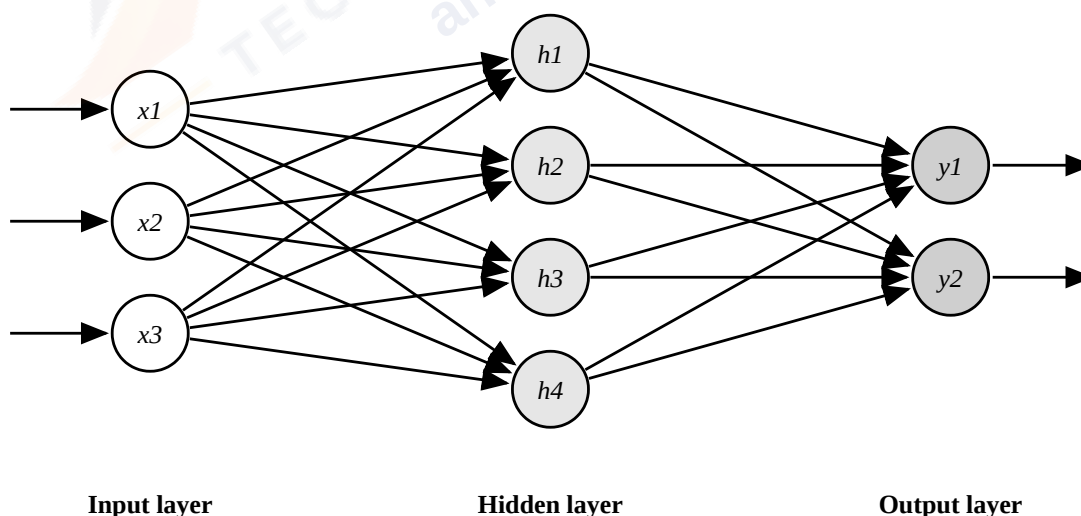


Fig.: Artificial neural network (input layer, hidden layer, output layer)

- **Input layer:** receives the input features (one neuron per feature); no computation.
- **Hidden layer(s):** each neuron computes the weighted sum of its inputs plus bias, $z = \sum w_i x_i + b$, and applies an **activation function** (step, sigmoid, ReLU, tanh) to give its output.
- **Output layer:** gives the final result (class or value).
- **Weights and bias** are learned during training by **back-propagation** with gradient descent to reduce the error between predicted and actual output.

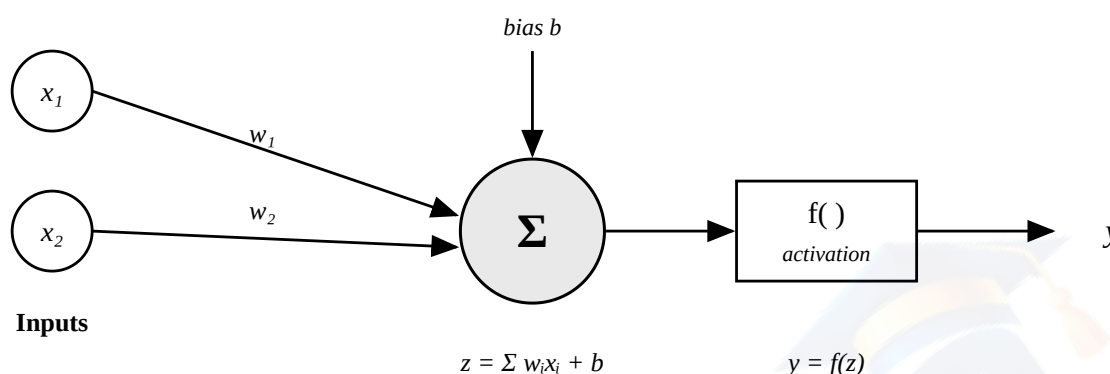


Fig.: Single neuron (perceptron) model

Numerical: $z = w_1 x_1 + w_2 x_2 + b = 0.6 \times 1 + 0.4 \times 1 + (-0.8) = 0.20$

(i) **Step function** ($y = 1$ if $z \geq 0$ else 0): $z = 0.20 \geq 0 \rightarrow y = 1$

(ii) **Sigmoid:** $y = 1/(1 + e^{-z}) = 1/(1 + e^{-0.20}) = 1/(1 + 0.8187) = 0.5498$

(Architecture with diagram 3 M, neuron model 1 M, numerical 2 M)

b) Multiple linear regression in Python.

6 M

Multiple linear regression predicts y from several inputs: $y = b_0 + b_1 x_1 + b_2 x_2 + \dots + b_n x_n$, where b_0 is the intercept and $b_1 \dots b_n$ are coefficients that show the effect of each feature (keeping the others constant). The coefficients are estimated by least squares.

Steps: 1) import libraries; 2) prepare X (features) and y (target); 3) create `LinearRegression()` and `fit(X, y)`; 4) read `intercept_` and `coef_`; 5) evaluate (R^2 , MSE) and predict new values.

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# features: [study hours per day, attendance %]    target: marks out of 100
X = np.array([[2, 60], [3, 65], [4, 70], [5, 75], [6, 80],
              [3, 85], [7, 90], [8, 95], [5, 60], [6, 92]])
y = np.array([45, 52, 60, 66, 73, 60, 84, 91, 61, 80])

model = LinearRegression().fit(X, y)
b0, (b1, b2) = model.intercept_, model.coef_
print(f"Equation: marks = {b0:.3f} + {b1:.3f}*hours + {b2:.3f}*attendance")
pred = model.predict(X)
print("R2 score:", round(r2_score(y, pred), 4), " MSE:",
      round(mean_squared_error(y, pred), 4))
print("Predicted marks (5 hours, 85% attendance):", round(model.predict([[5,
85]])[0], 2))
```

Output:

Equation: marks = 9.367 + 5.303*hours + 0.413*attendance

R2 score: 0.9981 MSE: 0.3607

Predicted marks (5 hours, 85% attendance): 70.95

(Concept and steps 2 M, program 3 M, output 1 M)

c) Multiclass confusion matrix and metrics.

6 M

For N classes the confusion matrix is an $N \times N$ table; row = actual class, column = predicted class. Diagonal cells are correct predictions; off-diagonal cells show which classes are confused. For each class (One-vs-Rest view): **TP** = diagonal cell, **FP** = column total – TP, **FN** = row total – TP.

Actual \ Predicted	Cat	Dog	Rabbit	Row total
Cat	15	3	2	20
Dog	2	18	0	20
Rabbit	1	2	17	20
Column total	18	23	19	60

Accuracy = $\Sigma \text{diagonal} / \text{total} = (15 + 18 + 17) / 60 = 50/60 = \mathbf{0.8333}$

Class	TP	FP	FN	Precision = $TP/(TP+FP)$	Recall = $TP/(TP+FN)$	F1 = $2PR/(P+R)$
Cat	15	3	5	$15/18 = 0.8333$	$15/20 = 0.7500$	0.7895
Dog	18	5	2	$18/23 = 0.7826$	$18/20 = 0.9000$	0.8372
Rabbit	17	2	3	$17/19 = 0.8947$	$17/20 = 0.8500$	0.8718

Macro-average F1 = $(0.7895 + 0.8372 + 0.8718) / 3 = \mathbf{0.8328}$. The model is best at recognising 'Rabbit' and confuses Cat with Dog most often.

(Explanation 1 M, accuracy 1 M, per-class metrics 4 M)

Q.6

a) Data cleaning — missing data and outliers.

6 M

Data cleaning is the process of finding and fixing errors, inconsistencies, missing values, duplicates and outliers in the dataset before training, because the quality of an ML model depends on the quality of its data.



Fig.: Data cleaning process

Handling missing data:

- **Deletion** — drop rows (listwise) or columns with many missing values (when the missing part is small).
- **Imputation** — fill with mean (numeric, no outliers), median (skewed data), mode (categorical), a constant, forward/backward fill, or a predicted value (k-NN / regression imputation).

Handling outliers:

- **Detect** — box plot / IQR rule (outside $Q1 - 1.5 \cdot IQR$, $Q3 + 1.5 \cdot IQR$), Z-score ($|z| > 3$), scatter plot.
- **Treat** — remove if it is an error, **cap** / **clip** to a limit (winsorising), transform (log), or keep if genuine.

Example (pandas):

```
import pandas as pd

df = pd.DataFrame({'City': ['Pune', 'Nanded', None, 'Pune', 'Latur', 'Pune', 'Nanded'],
                  'Salary': [25000, 27000, 26000, 25000, None, 250000, 28000]})
df = df.drop_duplicates() # remove repeated rows
df['City'] = df['City'].fillna(df['City'].mode()[0]) # categorical -> mode
df['Salary'] = df['Salary'].fillna(df['Salary'].median()) # numeric -> median

z = (df['Salary'] - df['Salary'].mean()) / df['Salary'].std()
print("Z-scores:", z.round(2).tolist())
Q1, Q3 = df['Salary'].quantile([0.25, 0.75])
upper = Q3 + 1.5 * (Q3 - Q1)
print("Upper IQR limit:", upper)
df['Salary'] = df['Salary'].clip(upper=upper) # cap the outlier
print(df.to_string(index=False))
```

Output:

```
Z-scores: [-0.43, -0.4, -0.41, -0.4, 2.04, -0.39]
Upper IQR limit: 30000.0
City Salary
Pune 25000.0
Nanded 27000.0
Nanded 26000.0
Latur 27000.0
Pune 30000.0
Nanded 28000.0
```

The duplicate row was removed, the missing city was filled with the mode (Pune), the missing salary with the median, and the outlier 250000 was capped at the upper IQR limit.

(Explanation 2 M, missing data 1½ M, outliers 1½ M, example 1 M)

b) Probabilistic reasoning using a Bayesian network.

6 M

Probabilistic reasoning uses probability to represent uncertain knowledge and to draw conclusions. A **Bayesian network** (belief network) is a **directed acyclic graph** in which nodes are random variables, arrows show direct (causal) influence, and every node has a **conditional probability table (CPT)** $P(X_i | \text{Parents}(X_i))$. The full joint distribution is $P(x_1, \dots, x_n) = \prod P(x_i | \text{Parents}(X_i))$.

Example: A house alarm (A) can be triggered by a burglary (B) or an earthquake (E). Neighbours John (J) and Mary (M) call when they hear the alarm.

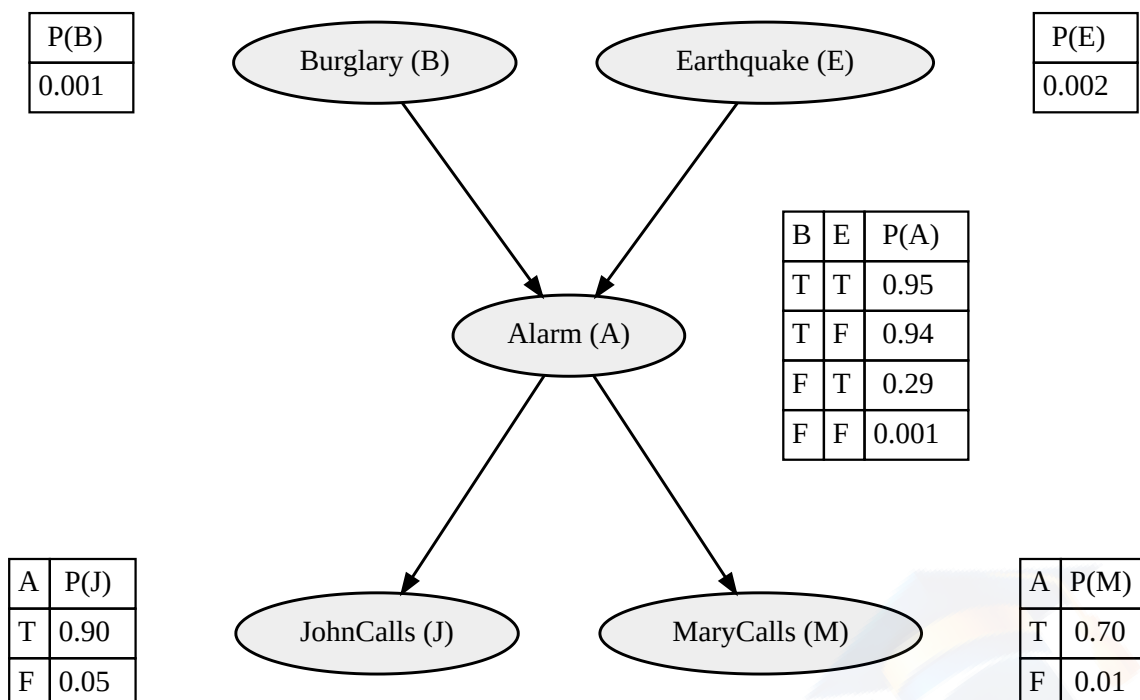


Fig.: Bayesian network for the burglary–alarm example (with CPTs)

Query 1: Probability that the alarm rings, both John and Mary call, but there is no burglary and no earthquake:

$$P(J, M, A, \neg B, \neg E) = P(J|A) \cdot P(M|A) \cdot P(A|\neg B, \neg E) \cdot P(\neg B) \cdot P(\neg E) = 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 = \mathbf{0.000628}$$

Query 2: If both John and Mary call, probability of burglary (by enumeration over E and A):

$$P(B, j, m) = \sum_{e,a} P(B)P(e)P(a|B,e)P(j|a)P(m|a) = 0.0005922; P(\neg B, j, m) = 0.0014919$$

$$P(B | j, m) = 0.0005922 / (0.0005922 + 0.0014919) = \mathbf{0.2842} \text{ (about 28\%)}$$

So the network lets the agent update its belief about the hidden cause (burglary) from observed evidence (calls) — this is probabilistic reasoning.

(Explanation 2 M, network with CPTs 2 M, calculation 2 M)

c) Supervised vs unsupervised; semi-supervised learning.

6 M

Point	Supervised learning	Unsupervised learning
Training data	Labelled (input–output pairs).	Unlabelled (inputs only).
Objective	Predict the correct output for new data.	Find structure, groups or rules in data.
Tasks	Classification, regression.	Clustering, association, dimensionality reduction.
Algorithms	Linear/logistic regression, k-NN, SVM, decision tree, random forest, Naive Bayes.	K-means, hierarchical clustering, DBSCAN, Apriori, PCA.
Evaluation	Easy — accuracy, precision, MSE etc. against labels.	Difficult — no ground truth (silhouette score etc.).
Cost	Labelling needs human experts — costly.	Raw data is cheap and plentiful.

Example	Predicting diabetes from medical records.	Segmenting mall customers by spending.
---------	---	--

Semi-supervised learning uses a **small amount of labelled data with a large amount of unlabelled data**. The model first learns from the labelled part and then labels the unlabelled data itself (self-training / pseudo-labelling), improving accuracy at low labelling cost. Example: classifying web pages or medical images where only a few are labelled by experts.

(Six points 4½ M, semi-supervised 1½ M)

Note: This is a **model question paper prepared by Ananda Technologies** only for practice and exam preparation, based on the pattern of previous MSBTE question papers and the K-scheme syllabus. It is **not an official MSBTE paper**, and it does **not** mean that these questions will come in the exam. Answers follow standard textbook points; marks shown are indicative. Free for students — anandatechnologies.in

