

AI & ML Algorithm (315330)

Model Question Paper — Set B · Model Answers

Q.1

a) Define AI; two applications.

2 M

Artificial Intelligence (AI) is the branch of computer science that makes machines (computers, robots, software) able to **think, learn and act intelligently like humans** — to perceive, reason, learn from experience, solve problems and take decisions.

Applications: 1) Medical diagnosis from images and reports; 2) Virtual assistants and chatbots (Alexa, Google Assistant); also self-driving cars, fraud detection, face recognition.

(Definition 1 M, two applications 1 M)

b) Properties of search algorithms.

2 M

- **Completeness** — the algorithm is guaranteed to find a solution if one exists.
- **Optimality** — the solution found has the lowest path cost among all solutions.
- **Time complexity** — time taken (number of nodes generated) to find a solution, e.g. $O(b^d)$.
- **Space complexity** — maximum memory (number of nodes stored) needed during the search.

(Four properties, ½ M each)

c) Why knowledge-based agents are used.

2 M

- They store knowledge about the world (facts and rules) in a **knowledge base** and use **reasoning** to take decisions, instead of fixed condition–action rules.
- They can **derive new knowledge** (inference) and act correctly in situations not explicitly programmed.
- New knowledge can be added easily by **TELL**, so the agent can learn and update itself; actions are asked using **ASK**.
- Useful for complex, partially observable environments, e.g. expert systems, diagnosis, Wumpus world.

(Any two reasons, 1 M each)

d) Data Science vs Data Analytics.

2 M

Data Science	Data Analytics
Broad, multi-disciplinary field: collects, processes and models data, builds ML / predictive algorithms.	Focused part of data science: analyses existing data to answer specific business questions.
Asks new questions; future-oriented (prediction, AI products).	Answers known questions; past/present oriented (reports, insights).
Tools: Python, R, TensorFlow, Hadoop, Spark.	Tools: Excel, SQL, Power BI, Tableau.

(Any two points, 1 M each)

e) Types of unsupervised learning algorithms.

2 M

- **Clustering** — K-means, hierarchical (agglomerative / divisive), DBSCAN.
- **Association rule learning** — Apriori, FP-Growth (market-basket analysis).
- **Dimensionality reduction** — Principal Component Analysis (PCA).
- **Anomaly detection**; neural-network based — autoencoders, self-organising maps.

(Any four, ½ M each)

f) Importance of confusion matrix.

2 M

A **confusion matrix** is an $N \times N$ table that compares the actual classes with the predicted classes of a classifier (TP, FP, FN, TN for two classes).

- It shows exactly where the model is confused — the **type of errors** (false positives and false negatives).
- Accuracy, precision, recall, specificity and F1-score are calculated from it; it is more informative than accuracy alone, especially for imbalanced data.

(Two points, 1 M each)

g) Linear regression.

2 M

Linear regression is a supervised learning algorithm that models a **linear relationship** between an independent variable x and a continuous dependent variable y : $y = a + bx$ (a = intercept, b = slope), fitted by minimising the sum of squared errors.

Example: predicting a student's marks from study hours, e.g. marks = $30 + 6.5 \times \text{hours}$; predicting house price from area.

(Definition 1 M, example 1 M)

Q.2

a) Breadth-First Search — trace.

4 M

BFS is an uninformed search that explores the tree **level by level**: all nodes at depth d are expanded before any node at depth $d+1$. It uses a **FIFO queue**. BFS is **complete**, and **optimal** when all step costs are equal; time and space complexity $O(b^d)$ (b = branching factor, d = depth of solution).

Algorithm: 1) Put start node in the queue and mark it visited. 2) Remove the front node; if it is the goal, stop. 3) Add all its unvisited neighbours at the rear of the queue (mark visited). 4) Repeat step 2 until the queue is empty.

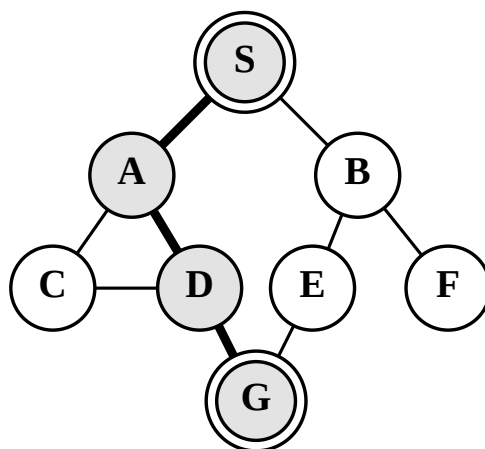


Fig.: Given graph (bold = path found by BFS)

Node removed	Unvisited neighbours added	Queue (front → rear)	Visited / expanded order
S	A, B	A, B	S
A	C, D	B, C, D	S, A
B	E, F	C, D, E, F	S, A, B
C	—	D, E, F	S, A, B, C
D	G	E, F, G	S, A, B, C, D
E	—	F, G	S, A, B, C, D, E
F	—	G	S, A, B, C, D, E, F
G	—	empty	S, A, B, C, D, E, F, G ✓ goal

BFS order: S, A, B, C, D, E, F, G. **Path to goal:** S → A → D → G (shallowest path, 3 edges).

(Explanation 1 M, trace 2 M, path 1 M)

b) Data cleaning — missing data and outliers (numerical).

4 M

Missing data are empty / NaN values caused by non-response, entry errors or faulty sensors.

Handling: delete rows/columns (if few), or **impute** with mean (normal data), **median** (skewed data / outliers present), mode (categorical), or predicted values.

Outliers are values that differ greatly from the rest. Detection: box plot / **IQR rule** (value < Q1 – 1.5·IQR or > Q3 + 1.5·IQR), Z-score ($|z| > 3$). Handling: remove, cap, or transform.

Step 1 — median of available values: sorted = 56, 58, 59, 60, 61, 62, 63, 98 (n = 8); median = (60 + 61)/2 = **60.50**. Filled data: 56, 61, 60.50, 58, 63, 60, 59, 98, 60.50, 62.

Step 2 — quartiles (sorted, n = 10): 56, 58, 59, 60, 60.50, 60.50, 61, 62, 63, 98

Q1 = (n+1)/4 th = 2.75th value = 58 + 0.75 × (59 – 58) = **58.75**

Q3 = 3(n+1)/4 th = 8.25th value = 62 + 0.25 × (63 – 62) = **62.25**

IQR = Q3 – Q1 = **3.50**; lower limit = Q1 – 1.5 × IQR = **53.50**; upper limit = Q3 + 1.5 × IQR = **67.50**

Outlier: 98 (outside the limits) — it should be checked and removed or capped. The median was used for filling because it is not affected by this outlier (the mean would be pulled up).

(Explanation 2 M, imputation 1 M, outlier 1 M)

c) Cross-validation; positive and negative class; significance.

4 M

Cross-validation is a resampling technique to estimate how well a model will perform on unseen data. The data is split into k folds; the model is trained on k-1 folds and tested on the remaining fold, repeated k times; the average score is reported.

	Fold 1	Fold 2	Fold 3	Fold 4
Iter 1	Test	Train	Train	Train
Iter 2	Train	Test	Train	Train
Iter 3	Train	Train	Test	Train
Iter 4	Train	Train	Train	Test

Final score = average of the k scores

Fig.: 4-fold cross-validation (shaded = test fold)

Positive class: the class we want to detect, coded 1 (e.g. 'fraud', 'diseased'). **Negative class:** the other class, coded 0 (e.g. 'genuine', 'healthy'). Correct and wrong predictions in each fold are counted as TP, TN, FP and FN with respect to the positive class; stratified folds keep the positive : negative ratio the same in every fold.

Significance:

- Gives a more reliable, less biased estimate of model performance than a single train-test split.
- Every record is used for both training and testing — useful when data is small.
- Detects overfitting and helps in model selection and hyper-parameter tuning (e.g. choosing k in k-NN).

(Explanation 1 M, positive/negative class 1 M, significance 2 M)

d) Metrics for classification — with calculation.

4 M

		Predicted	
		Disease	No disease
Actual	Disease	TP = 60	FN = 15
	No disease	FP = 20	TN = 105

- **Accuracy** = (TP + TN)/Total — overall correctness = $(60 + 105)/200 = 0.8250$
- **Precision** = $TP/(TP + FP)$ — how many predicted positives are correct = $60/80 = 0.7500$
- **Recall / Sensitivity** = $TP/(TP + FN)$ — how many actual positives are found = $60/75 = 0.8000$
- **Specificity** = $TN/(TN + FP)$ — how many actual negatives are found = $105/125 = 0.8400$
- **F1-score** = $2PR/(P + R)$ — harmonic mean of precision and recall = **0.7742**
- **ROC curve / AUC** — TPR vs FPR at different thresholds; AUC near 1 = good classifier.

(Metrics with formulas 2 M, calculations 2 M)

Q.3

a) Machine learning life cycle.

4 M

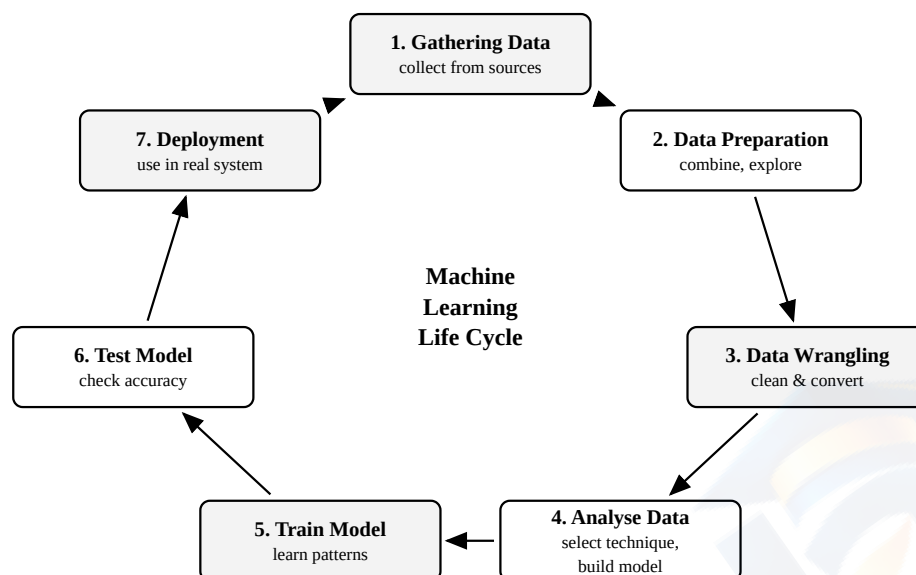


Fig.: Machine Learning life cycle (seven steps)

1. **Gathering data:** identify sources (files, databases, sensors, web) and collect data; quantity and quality decide model quality.
2. **Data preparation:** put data together, randomise it and explore it (understand features, format, quality).
3. **Data wrangling:** clean and convert raw data into usable format — handle missing values, duplicates, outliers, invalid data, noise.
4. **Analyse data:** select analytical technique (classification, regression, clustering...), build the model and review results.
5. **Train model:** feed training data to the algorithm so that it learns patterns and rules.
6. **Test model:** check accuracy of the trained model on the test dataset.
7. **Deployment:** put the model in the real system; monitor and improve it.

(Diagram 1 M, explanation 3 M)

b) Probabilistic reasoning; Bayes' theorem — numerical.

4 M

Need of probabilistic reasoning: In the real world an agent rarely has complete and certain knowledge — information may be incomplete, sensors noisy, rules may have exceptions and the world unpredictable. Probabilistic reasoning represents such **uncertain knowledge with probabilities** (degree of belief between 0 and 1) and updates beliefs when new evidence arrives, e.g. medical diagnosis, weather forecasting, spam filtering.

Bayes' theorem: $P(A|B) = P(B|A) \cdot P(A) / P(B)$

Given: $P(M1) = 0.6$, $P(M2) = 0.4$, $P(D|M1) = 0.03$, $P(D|M2) = 0.05$.

$P(D) = P(D|M1)P(M1) + P(D|M2)P(M2) = 0.03 \times 0.6 + 0.05 \times 0.4 = 0.018 + 0.020 = \mathbf{0.038}$

$$P(M2|D) = P(D|M2)P(M2) / P(D) = 0.020 / 0.038 = \mathbf{0.5263} \approx 52.63\%$$

(Need 1½ M, statement 1 M, numerical 1½ M)

c) Hill climbing — algorithm, problems and trace.

4 M

Hill climbing is a local search algorithm that starts from a (random) state and repeatedly moves to the **neighbouring state with a better value** of the objective function; it stops when no neighbour is better (a peak). It keeps only the current state (very little memory) and does not look back (greedy local search).

Steps: 1) Evaluate initial state. 2) Generate neighbours, pick the best. 3) If it is better than current, move to it; else stop and return current.

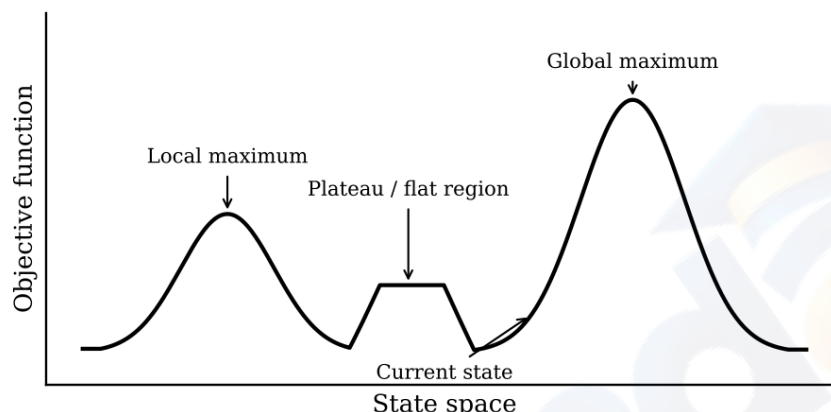


Fig.: State-space landscape for hill climbing

Problems: **Local maximum** (a peak lower than the global peak), **Plateau** (flat area — all neighbours equal), **Ridge** (higher area that cannot be reached by single moves). Remedies: random restart, backtracking, big jumps, simulated annealing.

Current state (value)	Neighbours (value)	Best neighbour	Action
1 (5)	0 (3), 2 (8)	2 (8)	move
2 (8)	1 (5), 3 (6)	3 (6)	stop — no better neighbour

Result: stops at state 2 with value 8. This is a **local maximum** — the global maximum is state 7 (value 12). Hill climbing got stuck; random restart could solve it.

(Explanation 1 M, problems 1 M, trace 1 M, comment 1 M)

d) Least-squares regression line — numerical.

4 M

x	y	x ²	xy
1	35	1	35
2	42	4	84
3	50	9	150
4	55	16	220
5	63	25	315
6	70	36	420
Σx = 21	Σy = 315	Σx ² = 91	Σxy = 1224

$$n = 6$$

$$b = [n\sum xy - \sum x \sum y] / [n\sum x^2 - (\sum x)^2] = [6 \times 1224 - 21 \times 315] / [6 \times 91 - 21^2] = 729 / 105 = 6.9429$$

$$a = \bar{y} - b \bar{x} = 52.5000 - 6.9429 \times 3.5000 = 28.2000$$

Regression line: $y = 28.2000 + 6.9429x$

Prediction for $x = 7$: $y = 28.2000 + 6.9429 \times 7 = 76.80$ marks

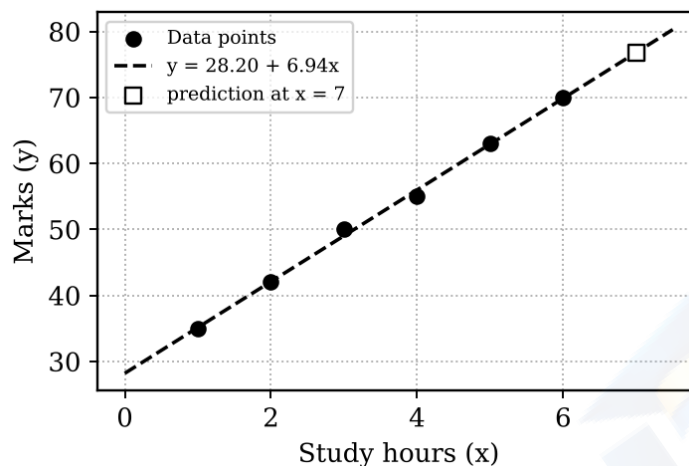


Fig.: Fitted regression line

(Table 1 M, b and a 2 M, prediction 1 M)

Q.4

a) Designing approaches for knowledge-based agents.

4 M

A knowledge-based agent can be built using two approaches:

- **Declarative approach:** start with an **empty knowledge base** and TELL the agent all the sentences (facts and rules) one by one until it knows how to work in the environment. Knowledge is written explicitly as logic sentences; the inference engine decides what to do. Easy to understand and modify.
- **Procedural approach:** the required behaviour is **directly encoded as program code** (procedures). Faster in execution, but harder to change and explain.
- In practice a successful agent **combines both**: declarative knowledge, compiled into efficient procedural code where needed.

Levels of a knowledge-based agent: 1) **Knowledge level** — what the agent knows and its goals (e.g. taxi knows the bridge connects A and B). 2) **Logical level** — how the knowledge is encoded as sentences of a logic. 3) **Implementation level** — physical representation of logic and knowledge (data structures, code).

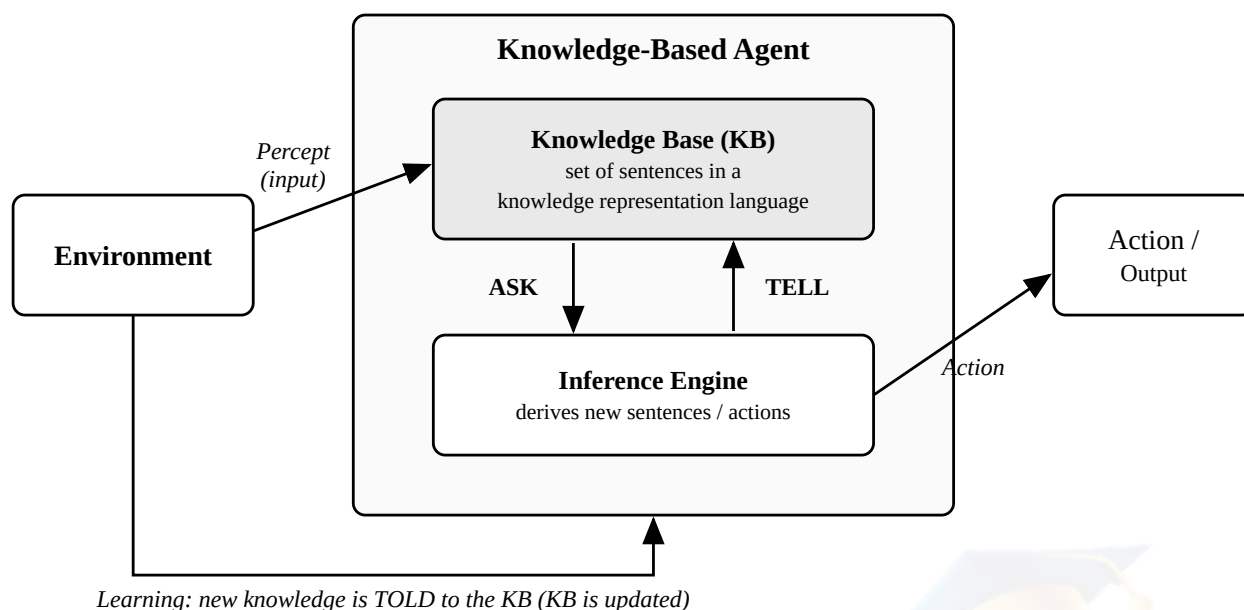


Fig.: Knowledge-based agent — KB with TELL / ASK interface

(Declarative 1½ M, procedural 1½ M, levels / diagram 1 M)

b) A* algorithm — trace.

4 M

A* expands the node with minimum $f(n) = g(n) + h(n)$ (g = cost so far, h = heuristic estimate to goal). OPEN = generated but not expanded nodes (sorted on f); CLOSED = expanded nodes.

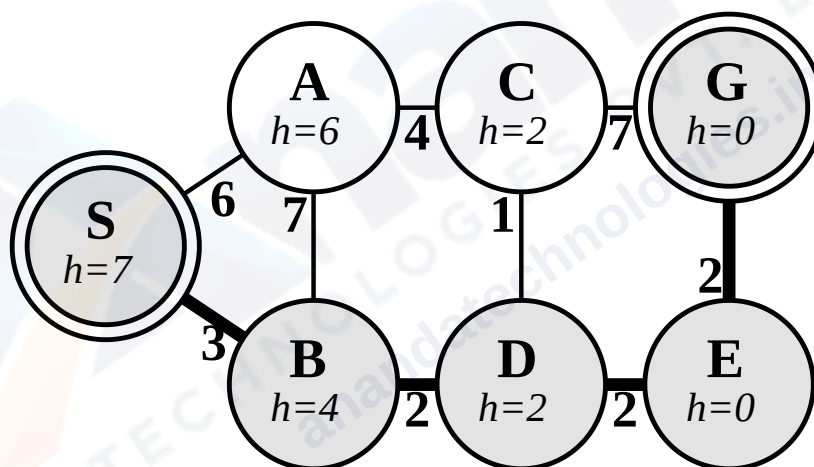


Fig.: Given graph; bold = optimal path found by A*

Step	Node selected (removed from Open)	Open list node($f = g + h$), sorted on f	Closed list
1	S ($g=0, h=7, f=7$)	B(7), A(12)	S
2	B ($g=3, h=4, f=7$)	D(7), A(12)	S, B
3	D ($g=5, h=2, f=7$)	E(7), C(8), A(12)	S, B, D
4	E ($g=7, h=0, f=7$)	C(8), G(9), A(12)	S, B, D, E
5	C ($g=6, h=2, f=8$)	G(9), A(12)	S, B, D, E, C
6	G ($g=9, h=0, f=9$)	Goal reached — stop	S, B, D, E, C, G

Optimal path: $S \rightarrow B \rightarrow D \rightarrow E \rightarrow G$; **total cost** = 9. (A* gives the optimal path because the given h never over-estimates the real cost — admissible.)

(Trace 3 M, path and cost 1 M)

c) Overfitting vs underfitting.

4 M

Overfitting	Underfitting
Model learns training data too well , including noise.	Model is too simple to learn the underlying pattern.
High training accuracy, low test accuracy.	Low accuracy on both training and test data.
Low bias, high variance .	High bias , low variance.
Cause: too complex model, too many features, small / noisy data, long training.	Cause: too simple model, too few features, insufficient training.
Remedy: more data, cross-validation, regularisation, pruning, early stopping, dropout.	Remedy: more complex model, more features, longer training, less regularisation.
Example: deep decision tree memorising every training record.	Example: straight line fitted to curved data.

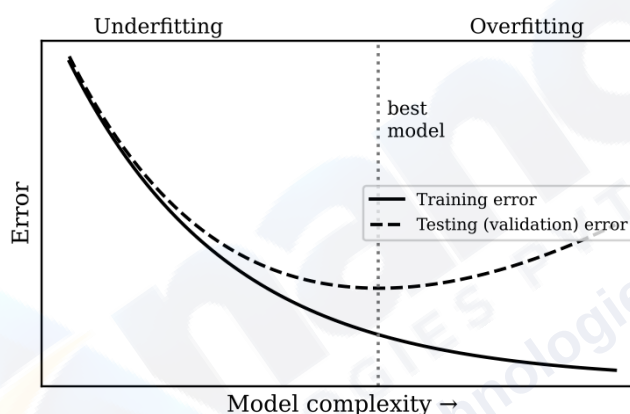


Fig.: Training and testing error vs model complexity

(Any four points, 1 M each)

d) k-NN classification — numerical.

4 M

k-NN: store all training records; for a new record compute its distance to every record, take the **k nearest** neighbours and assign the **majority class**.

Query point $Q = (162, 62)$; $d = \sqrt{[(h - 162)^2 + (w - 62)^2]}$

Height	Weight	Size	Distance from Q	Rank
158	58	M	$\sqrt{[(158-162)^2 + (58-62)^2]} = 5.66$	6
160	59	M	$\sqrt{[(160-162)^2 + (59-62)^2]} = 3.61$	3
163	61	M	$\sqrt{[(163-162)^2 + (61-62)^2]} = 1.41$	1
160	64	L	$\sqrt{[(160-162)^2 + (64-62)^2]} = 2.83$	2
165	64	L	$\sqrt{[(165-162)^2 + (64-62)^2]} = 3.61$	4
168	66	L	$\sqrt{[(168-162)^2 + (66-62)^2]} = 7.21$	7

170	68	L	$\sqrt{[(170-162)^2 + (68-62)^2]} = 10.00$	8
157	61	M	$\sqrt{[(157-162)^2 + (61-62)^2]} = 5.10$	5

3 nearest neighbours: (163, 61) → M [1.41], (160, 64) → L [2.83], (160, 59) → M [3.61]

Votes: M = 2, L = 1 → predicted size = **M**.

(Distances 2 M, nearest neighbours 1 M, prediction 1 M)

e) AI vs ML.

4 M

Artificial Intelligence	Machine Learning
Broad field of making machines that can think and act intelligently like humans.	A subset of AI in which machines learn from data without being explicitly programmed.
Goal: simulate human intelligence to solve complex problems; maximise chance of success.	Goal: learn from data to give accurate predictions; maximise accuracy.
Includes reasoning, knowledge representation, search, planning, NLP, robotics, ML.	Includes supervised, unsupervised and reinforcement learning.
Works with structured, semi-structured and unstructured data; can use rules or learning.	Works mainly with structured / semi-structured data; depends on data quality.
Examples: chess program, robots, expert systems, Siri.	Examples: spam filter, recommendation system, price prediction.

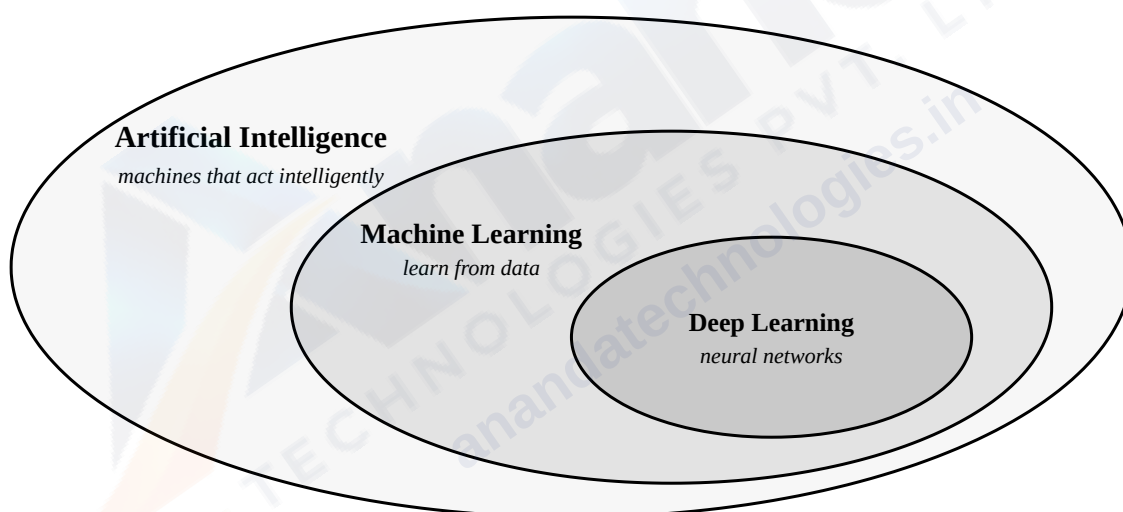


Fig.: AI ⊃ Machine Learning ⊃ Deep Learning

(Any four points, 1 M each)

Q.5

a) Random forest in Python; features.

6 M

Random forest is a supervised **ensemble** learning algorithm (bagging) that builds many decision trees on random bootstrap samples of the data, using a random subset of features at each split. For classification the final output is the **majority vote** of the trees; for regression it is the **average**.

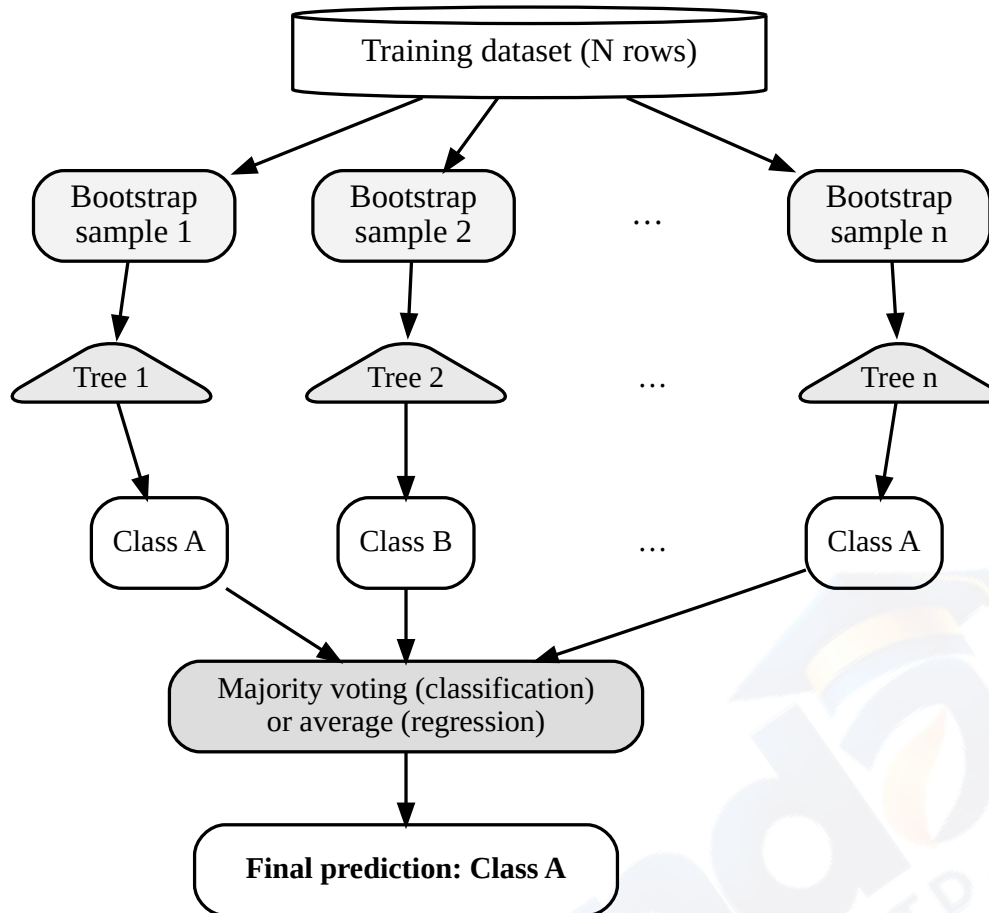


Fig.: Random forest — many decision trees on bootstrap samples, majority vote

```

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=1)

rf = RandomForestClassifier(n_estimators=100, random_state=1) # 100 trees
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

print("Accuracy:", round(accuracy_score(y_test, y_pred), 4))
print("Confusion matrix:")
print(confusion_matrix(y_test, y_pred))
for name, imp in zip(iris.feature_names, rf.feature_importances_):
    print(f"{name:20s} importance = {imp:.3f}")
sample = [[6.1, 2.9, 4.7, 1.4]]
print("Prediction for", sample[0], "->", iris.target_names[rf.predict(sample)
[0]])
  
```

Output:

```

Accuracy: 0.9556
Confusion matrix:
[[14  0  0]
  
```

```
[ 0 17  1]
[ 0  1 12]]
sepal length (cm)    importance = 0.121
sepal width (cm)     importance = 0.031
petal length (cm)    importance = 0.413
petal width (cm)     importance = 0.435
Prediction for [6.1, 2.9, 4.7, 1.4] -> versicolor
```

Features:

- Works for both classification and regression.
- Reduces overfitting compared to a single decision tree (averaging many trees) — high accuracy.
- Handles large datasets with many features; gives **feature importance**.
- Robust to noise, outliers and missing values; little data preparation needed.
- Trees are independent — can be trained in parallel (n_jobs).

(Explanation 1 M, program 3 M, features 2 M)

b) Working of training and testing data; significance.

6 M

In ML the available labelled dataset is divided into a **training set** (usually 70–80%) and a **testing set** (20–30%), often also a validation set.

- **Training data** is fed to the algorithm; the model adjusts its parameters (weights, splits, coefficients) to learn the relationship between inputs and outputs.
- **Testing data** is kept hidden during training. After training, the model predicts outputs for test inputs and the predictions are compared with actual labels to compute accuracy / error.
- The split must be **random** (and stratified for classification) so that both sets represent the same distribution; test data must never be used for training (data leakage).

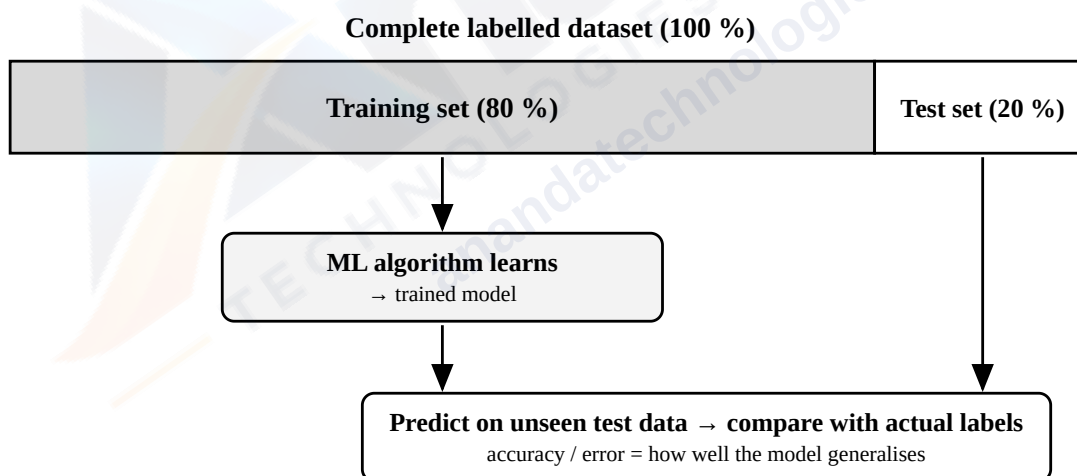


Fig.: Splitting a dataset into training (80%) and testing (20%) sets

Example (real output):

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

```
random_state=0)
print("Total records   :", len(X))
print("Training records:", len(X_train), " Testing records:", len(X_test))

model = DecisionTreeClassifier(random_state=0)
model.fit(X_train, y_train) # learning only from training data
print("Training accuracy:", round(model.score(X_train, y_train), 4))
print("Testing accuracy :", round(model.score(X_test, y_test), 4))
```

Output:

```
Total records   : 569
Training records: 455 Testing records: 114
Training accuracy: 1.0
Testing accuracy : 0.9123
```

Significance: training accuracy (1.0) is higher than testing accuracy — only the test set tells the true performance on unseen data; the gap shows overfitting. Testing data measures generalisation, helps compare models, tune hyper-parameters and decide whether the model is ready for deployment.

(Working 2 M, diagram 1 M, significance 2 M, example 1 M)

c) MAE, MSE, RMSE — explanation and numerical.

6 M

- **MAE (Mean Absolute Error)** = $(1/n) \sum |y_i - \hat{y}_i|$ — average size of errors, in the same unit as y ; treats all errors equally; robust to outliers.
- **MSE (Mean Squared Error)** = $(1/n) \sum (y_i - \hat{y}_i)^2$ — average of squared errors; penalises large errors heavily; unit is square of y ; differentiable, used as loss function.
- **RMSE (Root Mean Squared Error)** = $\sqrt{\text{MSE}}$ — same unit as y ; shows typical size of error while still giving more weight to big errors.
- For all three, **lower value = better model**; 0 means perfect prediction.

i	Actual y	Predicted $\hat{y}$	Error $e = y - \hat{y}$	e	e^2
1	50	48	2	2	4
2	62	65	-3	3	9
3	75	70	5	5	25
4	80	84	-4	4	16
5	95	92	3	3	9
Σ				17	63

MAE = $17/5 = 3.4000$; MSE = $63/5 = 12.6000$; RMSE = $\sqrt{12.6000} = 3.5496$ (₹ lakh)

(Three metrics 3 M, calculation 3 M)

Q.6

a) Supervised vs unsupervised learning.

6 M

Basis	Supervised learning	Unsupervised learning
Input data	Labelled data — each input has a known output.	Unlabelled data — only inputs.

Aim	Predict output for new inputs.	Discover hidden patterns / groups.
Feedback	Direct feedback — prediction compared with label.	No feedback.
Problems	Regression and classification.	Clustering and association.
Algorithms	Linear regression, logistic regression, k-NN, decision tree, random forest, SVM.	K-means, hierarchical clustering, Apriori, PCA.
Accuracy	Generally more accurate and measurable.	Less accurate; harder to measure.
Human effort	Labelling data is costly.	No labelling needed.
Example	Predicting whether a loan will default.	Grouping news articles by topic.

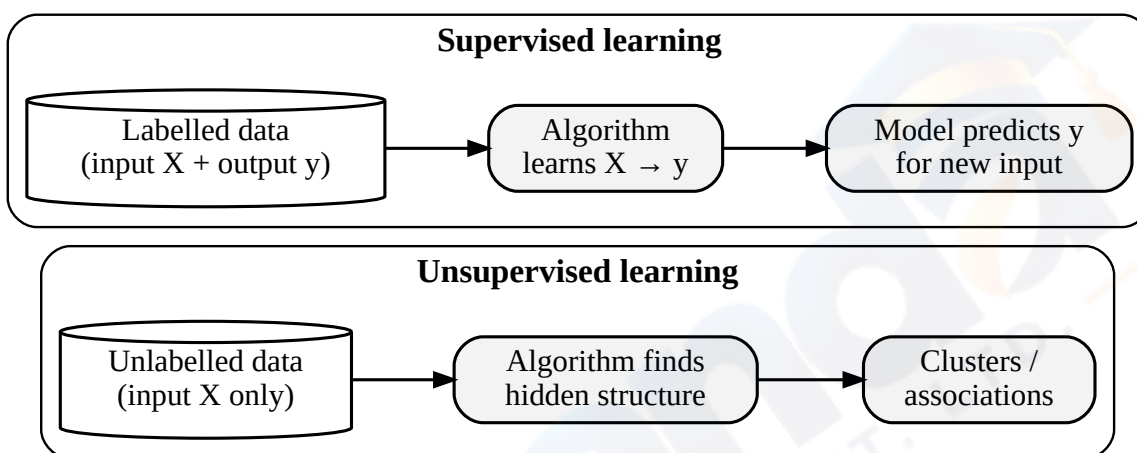


Fig.: Supervised vs unsupervised learning

(Any six points, 1 M each)

b) Components of AI.

6 M

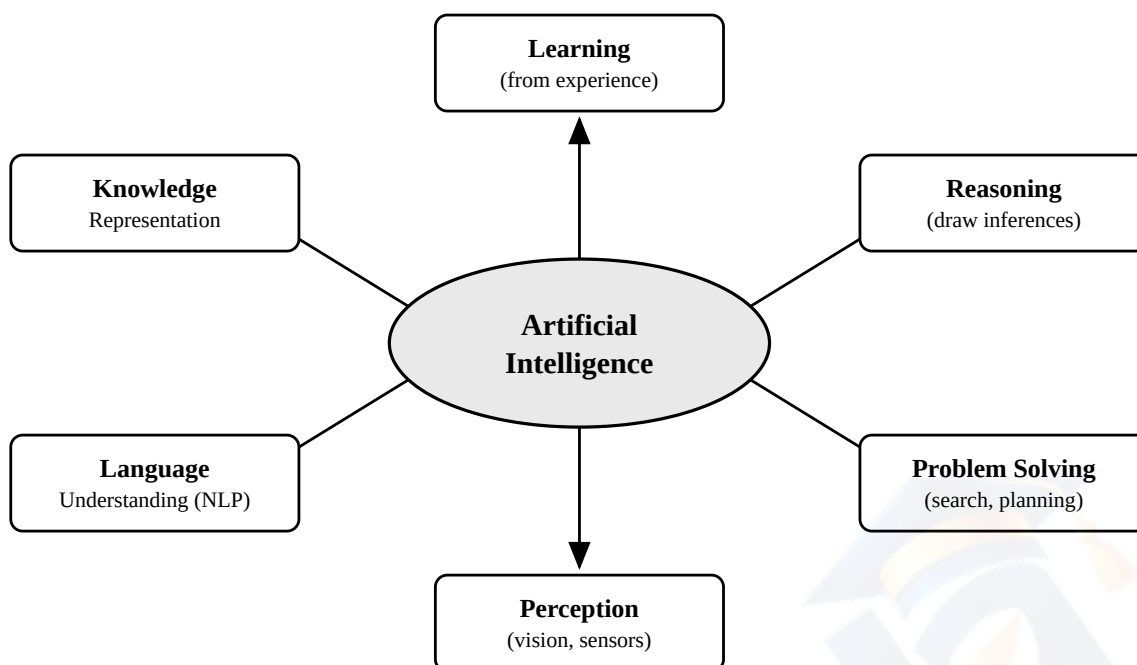


Fig.: Components of Artificial Intelligence

- **Learning:** improving performance from experience / data — rote learning, learning by examples, ML (supervised, unsupervised, reinforcement). E.g. a spam filter learns from marked mails.
- **Reasoning:** drawing conclusions (inferences) from facts and rules — deductive, inductive, abductive reasoning. E.g. 'All men are mortal; Ram is a man → Ram is mortal'.
- **Problem solving:** finding a sequence of actions from the initial state to the goal using search, planning and heuristics. E.g. route finding, puzzles, chess.
- **Perception:** acquiring and interpreting information from the environment through sensors — vision (camera), speech, touch. E.g. face recognition, self-driving car cameras.
- **Language understanding (NLP):** understanding and generating human language — speech recognition, translation, chatbots.
- **Knowledge representation:** storing facts about the world in a form the computer can reason with — logic, semantic networks, frames, rules.

(Diagram 1 M, any five components 1 M each)

c) Types of reasoning with examples.

6 M

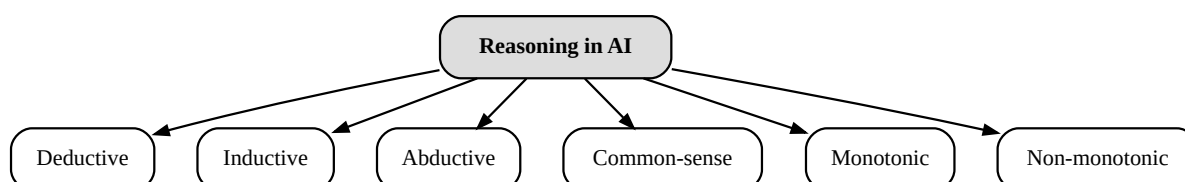


Fig.: Types of reasoning in AI

- **Deductive reasoning:** derives a **certain** conclusion from general premises (top-down). Ex: All birds have feathers; a sparrow is a bird → a sparrow has feathers.

- **Inductive reasoning:** generalises from specific observations (bottom-up); conclusion is probable. Ex: every swan I have seen is white → all swans are white.
- **Abductive reasoning:** starts from an observation and finds the **most likely explanation**. Ex: the road is wet → it probably rained. (Used in diagnosis.)
- **Common-sense reasoning:** informal reasoning using everyday experience. Ex: a person who puts his hand in fire gets burnt.
- **Monotonic reasoning:** once a conclusion is drawn it remains true even if new knowledge is added. Ex: Earth revolves around the Sun.
- **Non-monotonic reasoning:** conclusions may be withdrawn when new knowledge arrives. Ex: birds can fly → Tweety can fly; later 'Tweety is a penguin' → Tweety cannot fly.

(Diagram/list 1 M, five types with examples 5 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