Machine Learning-II(BAI702)
Internal Assessment Test 1– September 2025
Branch: AIML,CSE(AIML)

1 Define a well-posed learning problem. Explain the steps involved in designing a learning system.(5+5 M)

Answer-

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. (5 marks)

Example:

- Task (T): Playing checkers.
- Performance Measure (P): Win rate against human players.
- Experience (E): Playing games against itself or other players.
- As the program gains experience, its win rate improves → the learning problem is well-posed.

Steps in Designing a Learning System (5 Marks)

Designing a learning system involves the following key steps:

1. Specify the Task (T)
    - Define what the system is expected to do.
    - Example: Classify emails as spam or non-spam.
2. Select Performance Measure (P)
    - Determine how success will be measured.
    - Example: Accuracy of spam detection on unseen emails.

3. Specify the Training Experience (E)
    - Define the source of data or experience from which the system learns.
    - Example: Collection of labeled emails (spam/non-spam).
4. Choose Representation and Hypothesis Space
    - Decide how knowledge will be represented (e.g., decision trees, rules, neural networks).
    - Define the set of possible hypotheses the system can learn.
5. Design the Learning Algorithm
    - Select or create an algorithm that can generalize from experience to improve performance.
    - Example: Decision tree induction, k-NN, neural network training.
6. Training and Evaluation
    - Train the system using the experience.
    - Evaluate performance using the chosen measure and refine if necessary.

2 a Explain Version Spaces and the Candidate Elimination Algorithm. 5 Marks

Version Spaces (2 Marks)

- A version space is the set of all hypotheses in a hypothesis space that are consistent with the training examples.
- It is represented by two boundaries:
    1. S (Specific boundary): Most specific hypotheses consistent with the data.
    2. G (General boundary): Most general hypotheses consistent with the data.

- The hypotheses in the version space lie between S and G.
- Purpose: It helps to systematically represent all plausible hypotheses for a concept based on observed examples.

Candidate Elimination Algorithm (3 Marks)

- It is a learning algorithm that maintains the version space while processing examples.
- Goal: Narrow down the hypothesis space to include only hypotheses consistent with all observed examples.

Working Steps

1. Initialize:
    - S = most specific hypothesis (<Ø, Ø, Ø>),
    - G = most general hypothesis (<?, ?, ?>).
2. Process Each Training Example:
    - If positive example:
        - Remove hypotheses from G that do not classify it as positive.
        - Generalize hypotheses in S minimally to include the example.
    - If negative example:
        - Remove hypotheses from S that classify it as positive.
        - Specialize hypotheses in G minimally to exclude the example.
3. Update Version Space:
    - After each example, the S and G boundaries move closer.

- When S = G, the concept is fully learned.

2 b Apply the Find-S algorithm on a given dataset and write the final hypothesis.5 M

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Step 1: Initialize Hypothesis (3 Marks)

Find-S starts with the **most specific hypothesis**:

h = <Ø, Ø, Ø, Ø, Ø, Ø>

Step 2: Process Positive Examples Only

Positive examples are rows where EnjoySport = Yes (rows 1, 2, and 4).

Example 1: <Sunny, Warm, Normal, Strong, Warm, Same>

- Compare with h = <Ø, Ø, Ø, Ø, Ø, Ø>

Update hypothesis to **match the first positive example**:

h = <Sunny, Warm, Normal, Strong, Warm, Same>

Example 2: <Sunny, Warm, High, Strong, Warm, Same>

- Compare each attribute with current h:
    - Sky: Sunny = Sunny → keep Sunny
    - AirTemp: Warm = Warm → keep Warm
    - Humidity: Normal ≠ High → replace with ?
    - Wind: Strong = Strong → keep Strong
    - Water: Warm = Warm → keep Warm
    - Forecast: Same = Same → keep Same

Updated hypothesis:

h = <Sunny, Warm, ?, Strong, Warm, Same>

Example 3: Skip (EnjoySport = No)

- Negative example → Ignore in Find-S

Example 4: <Sunny, Warm, High, Strong, Cool, Change>

- Compare with current h:
    - Sky: Sunny = Sunny → keep Sunny
    - AirTemp: Warm = Warm → keep Warm
    - Humidity: ? → keep ?
    - Wind: Strong = Strong → keep Strong
    - Water: Warm ≠ Cool → replace with ?
    - Forecast: Same ≠ Change → replace with ?

Final hypothesis: (2 Marks)

h = <Sunny, Warm, ?, Strong, ?, ?>

3. Explain SOAR and PRODIGY. 10 M

SOAR (State, Operator, And Result)(2 Marks)

- Type: Cognitive architecture / Problem-solving system
- Purpose: General problem-solving and learning system for intelligent agents.
- Key Features: —(1 Mark)
    - State: Represents the current situation of the agent.
    - Operators: Actions that can change the state.
    - Decision Procedure: Chooses which operator to apply in a given state.
    - Learning: Uses chunking to learn from problem-solving experiences.
- Working:(1 Mark)
    - The system starts with an initial state and a goal.
    - It applies operators to move from the current state toward the goal.
    - Each experience is stored as a chunk (learned knowledge) to improve future problem-solving.

Example:(1Mark)

- SOAR can be used in robot navigation:
    - State: Robot at position X.
    - Operator: Move forward, turn left, pick object.

- Result: Updated robot state; learns best moves over time.

2. PRODIGY (PROblem DIrected Generation of hypotheses)(2 Marks)

- Type: Integrated learning and problem-solving system
- Purpose: Learns heuristics while solving problems to improve efficiency.
- Key Features:(1 Mark)
  - Combines planning and learning.
  - Learns from experience to guide future problem-solving.
  - Focuses on goal-directed learning (task-oriented).
- Working: (1 Mark)
  - PRODIGY starts with a goal and initial heuristics.
  - It tries to solve the problem using current heuristics.
  - If a solution fails or is inefficient, it learns new heuristics from the attempt.
- Example: (1 Mark)
  - Planning a sequence of actions to cook a recipe.
  - PRODIGY learns which sequences of actions are effective and applies this knowledge to similar recipes.

4 What is the Learn-One-Rule algorithm? Explain its working with an example. 10 M

Answer-

LEARN-ONE-RULE(*Target_attribute, Attributes, Examples, k*)

Returns a single rule that covers some of the Examples. Conducts a general_to_specific greedy beam search for the best rule, guided by the PERFORMANCE *metric.*

- Initialize *Best_hypothesis* to the most general hypothesis ∅
- Initialize *Candidate_hypotheses* to the set {*Best_hypothesis*}
- While *Candidate_hypotheses* is not empty, Do
  1. *Generate the next more specific candidate_hypotheses*
     - *All_constraints* ← the set of all constraints of the form $(a = v)$, where a is a member of *Attributes*, and v is a value of a that occurs in the current set of *Examples*
     - *New_candidate_hypotheses* ←
       for each h in *Candidate_hypotheses*,
       for each c in *All_constraints*,
       - create a specialization of h by adding the constraint c
     - Remove from *New_candidate_hypotheses* any hypotheses that are duplicates, inconsistent, or not maximally specific
  2. *Update Best_hypothesis*
     - For all h in *New_candidate_hypotheses* do
       - If (PERFORMANCE(h, *Examples*, *Target_attribute*)
         > PERFORMANCE(*Best_hypothesis*, *Examples*, *Target_attribute*))
         Then *Best_hypothesis* ← h
  3. *Update Candidate_hypotheses*
     - *Candidate_hypotheses* ← the k best members of *New_candidate_hypotheses*, according to the PERFORMANCE measure.
- Return a rule of the form
  "IF *Best_hypothesis* THEN *prediction*"
  where *prediction* is the most frequent value of *Target_attribute* among those *Examples* that match *Best_hypothesis*.

PERFORMANCE(h, *Examples*, *Target_attribute*)

- *h_examples* ← the subset of *Examples* that match h
- return −*Entropy*(*h_examples*), where entropy is with respect to *Target_attribute*

5 What are inductive–analytical approaches to learning? Explain with examples. 10 M (5 +5 M)

1 The Learning Problem—-5 M

Given:
A set of training examples D, possibly containing errors
A domain theory B, possibly containing errors
A space of candidate hypotheses H
Determine:
A hypothesis that best fits the training examples and domain theory

ErrorD(h) is defined to be the proportion of examples from D that are misclassified by h.

argmin kDerrorD (h) + kBerrorB (h)
h €H

.2 Hypothesis Space Search—----------5 M
How can the domain theory and training data best be combined to constrain the search for an acceptable hypothesis? This remains an open question in machine learning. This chapter surveys a variety of approaches that have been proposed,many of which consist of extensions to inductive methods we have already studied (e.g., BACKPROPAGATION, FOIL).One way to understand the range of possible approaches is to return to our view of learning as a task of searching through the space of alternative hypotheses.
We can characterize most learning methods as search algorithms by describing the hypothesis space H they search, the initial hypothesis ho at which they begin their search, the set of search operators 0 that define individual search steps, and the goal criterion G that specifies the search objective. In this chapter we explore three different methods for using prior knowledge to alter the search performed by purely inductive methods.

1.Use prior knowledge to derive an initial hypothesis from which to begin the Search.
In this approach the domain theory B is used to construct an initial hypothesis ho that is consistent with B. A standard inductive method is then applied, starting with the initial hypothesis ho.

2.Use prior knowledge to alter the objective of the hypothesis space search. In this approach, the goal criterion G is modified to require that the out- put hypothesis fits the domain theory as well as the training examples.

3. Use prior knowledge to alter the available search steps. In this approach, the set of search operators 0 is altered by the domain theory. For example, the FOCL system described below learns sets of Horn clauses in this way. It is based on the inductive system FOIL, which conducts a greedy search through the space of possible Horn clauses, at each step revising its current hypoth-
esis by adding a single new literal. FOCL uses the domain theory to expand the set of alternatives available when revising the hypothesis, allowing the addition of multiple literals in a single search step when warranted by the domain theory. In this way, FOCL allows single-step moves through the hypothesis space that would correspond to many steps using the original inductive algorithm. These "macro-moves" can dramatically alter the course of the search, so that the final hypothesis found consistent with the data is different from the one that would be found using only the inductive search

steps.

6.a) What is the primary purpose of using a Random Forest instead of a single decision tree? .2 M

Ans.The primary purpose of using a Random Forest instead of a single decision tree is to reduce overfitting and improve accuracy by combining the predictions of multiple decision trees through bagging and feature randomness.

6.b) Write a Python program to implement Bagging using DecisionTreeClassifier on a given dataset. Your program should:(2 M*4) 8 M

1. Load a dataset (e.g., Iris or any dataset of your choice).
2. Create a Bagging ensemble with 10 decision trees.
3. Train the ensemble on the training data.
4. Evaluate and print the accuracy on the test set

Answer.

```
# 1. Load dataset (Iris) (3 Mark)

iris = load_iris()
X, y = iris.data, iris.target

# Split into training and testing (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# 2. Create a Bagging ensemble with 10 Decision Trees ( 5 Mark)

base_dt = DecisionTreeClassifier(random_state=42)
bagging_model = BaggingClassifier(
    base_estimator=base_dt,
    n_estimators=10,      # number of decision trees
    random_state=42
)

# 3. Train the ensemble
```

```
bagging_model.fit(X_train, y_train)

# 4. Evaluate accuracy on test set
y_pred = bagging_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy of Bagging with Decision Trees:", accuracy)
```