Lab Questions:

1. Write a short Python script to simulate a rule-based chatbot using if-else logic.
2. Write Python script that simulates both a simple reflex agent and a model-based reflex agent for a vacuum cleaner
3. Solve 8-puzzle problem using BFS, DFS.
4. Implement A* on a grid map with heuristic.
5. Write a Tic-Tac-Toe game with a mini-max AI player. (optional: Use alpha-beta pruning for optimization.)
6. Implement forward chaining and backward chaining rule-based systems.
7. Solve medical diagnosis using Bayes Rule.
8. Implement Naive Bayes classifier on text (spam classification).
   OR
   Text classification using Naive Bayes.
9. Perform clustering on Iris dataset using sklearn.
10. Solve the Traveling Salesman Problem (TSP) using Genetic Algorithm in Python.
11. Build a perceptron for binary classification.
12. Train a simple ANN using TensorFlow or Keras on MNIST.
13. Tokenize and POS-tag a sentence using spaCy or nltk
14. Classify images using a pretrained CNN (MobileNetV2 with TensorFlow).

Answer:

✅ **1. Rule-Based Chatbot Using If-Else**

```
def chatbot():
    print("Chatbot: Hi! I'm a simple chatbot. Type 'bye' to exit.")
    while True:
        user_input = input("You: ").lower()

        if user_input == "bye":
            print("Chatbot: Goodbye! Have a nice day.")
            break
        elif "hello" in user_input or "hi" in user_input:
            print("Chatbot: Hello there! How can I help you?")
        elif "how are you" in user_input:
            print("Chatbot: I'm just a bunch of code, but I'm doing great!")
        elif "name" in user_input:
            print("Chatbot: I'm called RuleBot.")
        elif "help" in user_input:
            print("Chatbot: I can respond to greetings, tell you my name, and chat a bit.")
        else:
            print("Chatbot: Sorry, I don't understand that.")

# Run the chatbot
chatbot()
```

✅ **2. Simple Reflex Agent vs Model-Based Reflex Agent (Vacuum Cleaner)**

💡 **Environment:**

- Two locations: A and B
- Each can be Dirty (1) or Clean (0)

🧠 **Simple Reflex Agent:**

- Makes decisions based **only on current status**.

🧠 **Model-Based Reflex Agent:**

- **Remembers** previous states.
- Stops (NoOp) if both are clean.

```
# 2. Reflex and Model-based Vacuum Cleaner Agent
def simple_reflex_agent(location, status):
    if status == "dirty":
        return "Clean"
    elif location == "A":
        return "Move Right"
    else:
        return "Move Left"

def model_based_reflex_agent(location, status, model):
```

```
    model[location] = status
    if status == "dirty":
        return "Clean"
    elif model['A'] == "clean" and model['B'] == "clean":
        return "Stop"
    elif location == "A":
        return "Move Right"
    else:
        return "Move Left"

# Example usage:
# model = {'A': None, 'B': None}
# print(model_based_reflex_agent('A', 'clean', model))
```

🧩 **8-Puzzle Problem**

- A 3×3 grid puzzle with tiles 1 to 8 and a blank 0.
- Goal: Arrange tiles in order 123456780.

✅ **BFS (Breadth-First Search)**

- Explores states level by level.
- Guarantees shortest solution.
- **BFS Path (Steps to Goal):**

```
['123405678', '123450678', '123458670', '123458607', '123458067',
 '123058467', '123508467', '123568407', '123568470', '123560478',
 '123506478', '123056478', '123456078', '123456708', '123456780']
```

- ✅ **Solved** in 14 steps!

❌ **DFS (Depth-First Search)**

- Goes deep along one branch before backtracking.
- May miss solution without depth limit or optimal conditions.
- **Result**: None (didn't find solution in the depth limit of 50)

🗺️ **Grid Environment:**

0 = Free space
1 = Obstacle

Grid:
[0, 0, 0, 0, 0]
[0, 1, 1, 1, 0]
[0, 0, 0, 1, 0]
[1, 1, 0, 0, 0]

✅ **Start: (0, 0)**

✅ **Goal: (3, 4)**

✅ **Heuristic: Manhattan distance**

```
# 3. 8-Puzzle using BFS and DFS
from collections import deque

def is_goal(state):
    return state == [1, 2, 3, 4, 5, 6, 7, 8, 0]

def get_neighbors(state):
    moves = []
    idx = state.index(0)
    row, col = divmod(idx, 3)
    directions = {'up': -3, 'down': 3, 'left': -1, 'right': 1}
    for move, delta in directions.items():
        new_idx = idx + delta
        if move == 'left' and col == 0 or move == 'right' and col == 2:
            continue
        if 0 <= new_idx < 9:
            new_state = state[:]
            new_state[idx], new_state[new_idx] = new_state[new_idx], new_state[idx]
            moves.append(new_state)
    return moves

def bfs(start):
    visited = set()
    queue = deque([[start]])
    while queue:
        path = queue.popleft()
        state = path[-1]
        if tuple(state) in visited:
            continue
        visited.add(tuple(state))
        if is_goal(state):
            return path
        for neighbor in get_neighbors(state):
            queue.append(path + [neighbor])
    return None

def dfs(start):
    visited = set()
    stack = [[start]]
    while stack:
        path = stack.pop()
        state = path[-1]
```

```
        if tuple(state) in visited:
            continue
        visited.add(tuple(state))
        if is_goal(state):
            return path
        for neighbor in get_neighbors(state):
            stack.append(path + [neighbor])
    return None

# Example: start = [1, 2, 3, 4, 0, 5, 6, 7, 8]
```

🧠 **A* Algorithm**

- Combines **path cost** (g) and **heuristic** (h) for efficient search.
- Always finds the **shortest path** if one exists.

📍 **Path Found:**

[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (2, 4), (3, 4)]

✔ Short and valid path avoiding obstacles!

```
# 4. A* on a Grid Map
import heapq

def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def astar(grid, start, goal):
    rows, cols = len(grid), len(grid[0])
    open_set = [(0 + heuristic(start, goal), 0, start)]
    came_from = {}
    g_score = {start: 0}
    while open_set:
        _, cost, current = heapq.heappop(open_set)
        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            return path[::-1]
        for dx, dy in [(-1,0), (1,0), (0,-1), (0,1)]:
            neighbor = (current[0]+dx, current[1]+dy)
            if 0 <= neighbor[0] < rows and 0 <= neighbor[1] < cols:
                if grid[neighbor[0]][neighbor[1]] == 1:
                    continue
                tentative_g = g_score[current] + 1
                if neighbor not in g_score or tentative_g < g_score[neighbor]:
                    g_score[neighbor] = tentative_g
```

```
            f = tentative_g + heuristic(neighbor, goal)
            heapq.heappush(open_set, (f, tentative_g, neighbor))
            came_from[neighbor] = current
    return None
```

✅ 5. Tic-Tac-Toe with Minimax AI

```
def print_board(board):
    for i in range(0, 9, 3):
        print(board[i] + " | " + board[i+1] + " | " + board[i+2])
        if i < 6:
            print("--+---+--")

def check_winner(board, player):
    win_combos = [
        [0,1,2], [3,4,5], [6,7,8],
        [0,3,6], [1,4,7], [2,5,8],
        [0,4,8], [2,4,6]
    ]
    return any(all(board[i] == player for i in combo) for combo in win_combos)

def is_draw(board):
    return all(cell != " " for cell in board)

def minimax(board, is_max):
    if check_winner(board, "O"): return 1
    if check_winner(board, "X"): return -1
    if is_draw(board): return 0

    best_score = -float('inf') if is_max else float('inf')
    for i in range(9):
        if board[i] == " ":
            board[i] = "O" if is_max else "X"
            score = minimax(board, not is_max)
            board[i] = " "
            best_score = max(score, best_score) if is_max else min(score, best_score)
    return best_score

def best_move(board):
    best_score = -float("inf")
    move = -1
    for i in range(9):
        if board[i] == " ":
            board[i] = "O"
            score = minimax(board, False)
            board[i] = " "
```

```
            if score > best_score:
                best_score = score
                move = i
    return move

def play_game():
    board = [" "] * 9
    print("You are X. AI is O.")
    print_board(board)

    while True:
        # Human
        while True:
            try:
                pos = int(input("Enter position (1-9): ")) - 1
                if board[pos] == " ":
                    board[pos] = "X"
                    break
            except:
                pass
            print("Invalid move. Try again.")

        print_board(board)
        if check_winner(board, "X"):
            print("You win!")
            break
        if is_draw(board):
            print("It's a draw!")
            break

        # AI
        ai = best_move(board)
        board[ai] = "O"
        print("AI played:")
        print_board(board)
        if check_winner(board, "O"):
            print("AI wins!")
            break
        if is_draw(board):
            print("It's a draw!")
            break

# Uncomment below to run:
# play_game()
```

✅ 6. Forward and Backward Chaining Rule-Based System

```
# Sample knowledge base (rules and facts)
rules = [
    {"if": ["A", "B"], "then": "C"},
    {"if": ["C", "D"], "then": "E"},
    {"if": ["E"], "then": "F"},
]

# Initial known facts
facts = set(["A", "B", "D"])

# Forward Chaining
def forward_chaining(rules, facts):
    inferred = set()
    changed = True
    while changed:
        changed = False
        for rule in rules:
            if all(cond in facts for cond in rule["if"]) and rule["then"] not in facts:
                facts.add(rule["then"])
                inferred.add(rule["then"])
                changed = True
    return inferred

# Backward Chaining
def backward_chaining(rules, facts, goal):
    if goal in facts:
        return True
    for rule in rules:
        if rule["then"] == goal:
            if all(backward_chaining(rules, facts, cond) for cond in rule["if"]):
                return True
    return False

# Run
inferred_facts = forward_chaining(rules, set(facts))
goal_to_prove = "F"
is_proved = backward_chaining(rules, set(facts), goal_to_prove)

print("Inferred facts (Forward Chaining):", inferred_facts)
print(f"Is '{goal_to_prove}' provable? (Backward Chaining):", is_proved)
```

✅ 7. Medical Diagnosis using Bayes' Theorem

```
# Problem:
# - Disease probability P(D) = 0.01
```

```
# - Test accuracy:
#   - True positive: P(Pos | D) = 0.99
#   - False positive: P(Pos | ~D) = 0.05

# Bayes' Theorem:
# P(D | Pos) = (P(Pos|D) * P(D)) / [P(Pos|D)*P(D) + P(Pos|~D)*P(~D)]

P_D = 0.01              # Prior probability of disease
P_not_D = 1 - P_D       # Probability of no disease
P_Pos_given_D = 0.99    # True positive rate
P_Pos_given_not_D = 0.05  # False positive rate

# Apply Bayes' Rule
P_D_given_Pos = (
    P_Pos_given_D * P_D
) / (
    P_Pos_given_D * P_D + P_Pos_given_not_D * P_not_D
)

print(f"Probability of having disease given positive test: {P_D_given_Pos:.4f}")
```

✅ 8. Naive Bayes Classifier on Text (Spam Detection)

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

# Sample text data
texts = [
    "Win money now",            # spam
    "Limited time offer",       # spam
    "Call now and win big",     # spam
    "Meeting at 10am",          # ham
    "Project deadline tomorrow", # ham
    "Let's have lunch",         # ham
]
labels = ["spam", "spam", "spam", "ham", "ham", "ham"]

# Convert text to bag-of-words vectors
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)

# Train the Naive Bayes classifier
model = MultinomialNB()
model.fit(X, labels)

# Test new messages
```

```
test_texts = ["Win a free lunch now", "Lunch meeting tomorrow"]
X_test = vectorizer.transform(test_texts)
predictions = model.predict(X_test)

# Output
for msg, label in zip(test_texts, predictions):
    print(f"'{msg}' => {label}")
```

✅ 9. Clustering on the Iris Dataset using sklearn (K-Means)

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target  # actual species (not used in clustering)

# KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(X)
labels = kmeans.labels_

# Plot the clusters using first two features
plt.figure(figsize=(8, 5))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o', edgecolor='k')
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.title("KMeans Clustering on Iris Dataset (First 2 Features)")
plt.grid(True)
plt.show()
```

🧠 Notes:

- load_iris() loads 150 flower samples from 3 species.
- KMeans(n_clusters=3) groups the data into 3 clusters.
- We're visualizing only the **first two features** (Sepal length & width).

✅ 10. TSP using Genetic Algorithm in Python

```
import random
import numpy as np

# Cities and coordinates
cities = {
    "A": (0, 0),
    "B": (2, 3),
```

```
    "C": (5, 4),
    "D": (1, 6),
    "E": (7, 2)
}
city_names = list(cities.keys())

# Helper: distance between cities
def distance(a, b):
    xa, ya = cities[a]
    xb, yb = cities[b]
    return np.hypot(xa - xb, ya - yb)

# Total path distance
def total_distance(path):
    return sum(distance(path[i], path[i+1]) for i in range(len(path)-1)) + distance(path[-1], path[0])

# Initial population
def create_population(size):
    return [random.sample(city_names, len(city_names)) for _ in range(size)]

# Crossover: ordered crossover
def crossover(p1, p2):
    start, end = sorted(random.sample(range(len(p1)), 2))
    child = p1[start:end+1]
    child += [c for c in p2 if c not in child]
    return child

# Mutation: swap two cities
def mutate(path, rate=0.1):
    if random.random() < rate:
        i, j = random.sample(range(len(path)), 2)
        path[i], path[j] = path[j], path[i]
    return path

# Main Genetic Algorithm
def genetic_algorithm(generations=100, pop_size=20):
    population = create_population(pop_size)
    for _ in range(generations):
        population.sort(key=total_distance)
        next_gen = population[:2]  # keep 2 best
        while len(next_gen) < pop_size:
            p1, p2 = random.choices(population[:5], k=2)
            child = crossover(p1, p2)
            next_gen.append(mutate(child))
        population = next_gen
```

```
    best = min(population, key=total_distance)
    print("Best path:", best)
    print("Total distance:", total_distance(best))

# Run
genetic_algorithm()
```

✅ 11. Perceptron for Binary Classification

```
import numpy as np

class Perceptron:
    def __init__(self, learning_rate=0.1, n_iters=10):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        # Initialize weights and bias
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self._activation(linear_output)

                # Update rule
                update = self.lr * (y[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        y_predicted = self._activation(linear_output)
        return y_predicted

    def _activation(self, x):
        return np.where(x >= 0, 1, 0)

# Example usage:
if __name__ == "__main__":
    # Simple dataset (AND logic gate)
    X = np.array([[0,0], [0,1], [1,0], [1,1]])
```

```
    y = np.array([0, 0, 0, 1])  # AND outputs

    p = Perceptron(learning_rate=0.1, n_iters=10)
    p.fit(X, y)

    print("Weights:", p.weights)
    print("Bias:", p.bias)

    # Predictions
    preds = p.predict(X)
    print("Predictions:", preds)
```

How it works:

- The perceptron learns weights to classify inputs into two classes (0 or 1).
- Here, it learns the AND logic gate as an example.

Try running the code! You can replace X and y with your own binary data.

✅ **12. Train a Simple ANN on MNIST with TensorFlow/Keras**

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load MNIST data
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Normalize pixel values (0 to 1)
X_train = X_train / 255.0
X_test = X_test / 255.0

# One-hot encode labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Build model
model = Sequential([
    Flatten(input_shape=(28, 28)),       # Flatten 28x28 images to 784 vector
    Dense(128, activation='relu'),       # Hidden layer with 128 neurons
    Dense(10, activation='softmax')      # Output layer for 10 classes
])

# Compile model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
```

```
              metrics=['accuracy'])

# Train model
model.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.2)

# Evaluate on test data
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {accuracy:.4f}")
```

What's happening?

- MNIST digits loaded and normalized
- Labels one-hot encoded for 10 classes (digits 0-9)
- A simple feedforward ANN with one hidden layer (128 neurons)
- Trained for 5 epochs, then evaluated on test set

Run this code in a Python environment with TensorFlow installed (pip install tensorflow)!

✅ 13. Tokenize and POS-tag a sentence

Using NLTK

```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

sentence = "ChatGPT is an amazing AI assistant."

# Tokenize sentence into words
tokens = nltk.word_tokenize(sentence)

# POS tagging
pos_tags = nltk.pos_tag(tokens)

print("Tokens:", tokens)
print("POS tags:", pos_tags)
```

Using spaCy

```
import spacy

# Load English model (run once)
nlp = spacy.load("en_core_web_sm")

sentence = "ChatGPT is an amazing AI assistant."

doc = nlp(sentence)

tokens = [token.text for token in doc]
```

```
pos_tags = [(token.text, token.pos_) for token in doc]

print("Tokens:", tokens)
print("POS tags:", pos_tags)
```

Notes:

- For NLTK, you may need to run nltk.download() for necessary data.
- For spaCy, you may need to install the model first with:
  python -m spacy download en_core_web_sm

✅ **14. Image Classification with Pretrained MobileNetV2 (TensorFlow)**

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt

# Load pretrained MobileNetV2 model + weights
model = MobileNetV2(weights='imagenet')

# Load and preprocess an image
img_path = 'elephant.jpg'  # Replace with your image file path
img = image.load_img(img_path, target_size=(224, 224))  # MobileNetV2 expects 224x224

# Display image
plt.imshow(img)
plt.axis('off')
plt.show()

# Convert image to array and preprocess
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

# Predict
preds = model.predict(x)

# Decode prediction results
print('Predicted:', decode_predictions(preds, top=3)[0])
```

How to run:

1. Install required packages if not installed:
   pip install tensorflow matplotlib
2. Place an image (e.g., elephant.jpg) in your working directory or update img_path to your image.

3. Run the script.

This will show the image and print the top 3 predicted labels with probabilities.