YUVANESH KS

AI&ML LAB Answer Batch 1&2

Batch 1 :

---

1. Write a python program to implement BFS. The graph is implemented as an adjacency list.

```
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])

    while queue:
        node = queue.popleft()
        if node not in visited:
            print(node, end=" ")
            visited.add(node)
            queue.extend([n for n in graph[node] if n not in visited])

# Example usage:
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [], 'E': [], 'F': []
}
bfs(graph, 'A')
```

---

2. Write a python program to implement DFS. The graph is implemented as an adjacency list.

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start, end=" ")
    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)

# Example usage:
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [], 'E': [], 'F': []
}
dfs(graph, 'A')
```

---

*3. Write a python program to implement informed search algorithms using A algorithm.**

```
from queue import PriorityQueue

def a_star(graph, start, goal, h):
    open_list = PriorityQueue()
    open_list.put((0, start))
    came_from = {}
    g_score = {start: 0}

    while not open_list.empty():
        _, current = open_list.get()

        if current == goal:
            path = []
            while current in came_from:
                path.insert(0, current)
                current = came_from[current]
            path.insert(0, start)
            return path

        for neighbor, cost in graph[current]:
            tentative_g = g_score[current] + cost
            if neighbor not in g_score or tentative_g < g_score[neighbor]:
                g_score[neighbor] = tentative_g
                f = tentative_g + h[neighbor]
                open_list.put((f, neighbor))
                came_from[neighbor] = current

    return None

# Example usage:
graph = {
    'A': [('B', 1), ('C', 4)],
    'B': [('D', 1)],
    'C': [('D', 2)],
    'D': []
}
h = {'A': 4, 'B': 2, 'C': 1, 'D': 0}
print(a_star(graph, 'A', 'D', h))
```

---

*4. Write a python program to implement memory-bounded A algorithm. (Simplified SMA)***

```
# Memory-bounded A* is complex; below is a simplified simulation of Limited A*

def limited_a_star(graph, start, goal, h, limit):
    from queue import PriorityQueue

    class Node:
        def __init__(self, state, path, cost):
            self.state = state
            self.path = path
            self.cost = cost

        def __lt__(self, other):
```

```python
            return self.cost < other.cost

    frontier = PriorityQueue()
    frontier.put((h[start], Node(start, [start], 0)))

    while not frontier.empty() and frontier.qsize() <= limit:
        _, current = frontier.get()
        if current.state == goal:
            return current.path

        for neighbor, weight in graph.get(current.state, []):
            g = current.cost + weight
            f = g + h[neighbor]
            frontier.put((f, Node(neighbor, current.path + [neighbor], g)))

    return None

# Example usage:
graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('D', 1)],
    'C': [('D', 1)],
    'D': []
}
h = {'A': 3, 'B': 2, 'C': 1, 'D': 0}
print(limited_a_star(graph, 'A', 'D', h, limit=3))
```

---

5. Write a python program to implement Gaussian Naive Bayes models.

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

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

model = GaussianNB()
model.fit(X_train, y_train)

print("Accuracy:", model.score(X_test, y_test))
```

---

6. Write a python program to implement Bernoulli Naive Bayes models.

```python
from sklearn.naive_bayes import BernoulliNB
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

X, y = make_classification(n_samples=100, n_features=10, random_state=42)
X = (X > 0).astype(int)  # Binarize
X_train, X_test, y_train, y_test = train_test_split(X, y)

model = BernoulliNB()
model.fit(X_train, y_train)
```

```
print("Accuracy:", model.score(X_test, y_test))
```

---

7. Write a python program to implement Multinomial Naive Bayes models.

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

docs = ["I love AI", "AI is fun", "Machine Learning is part of AI"]
labels = [1, 1, 0]

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(docs)

model = MultinomialNB()
model.fit(X, labels)

print("Prediction for 'AI rocks':", model.predict(vectorizer.transform(["AI
rocks"])))
```

---

8. Write a program to construct a Bayesian network for medical data and use it to diagnose heart patients.

```
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination

# Define structure
model = BayesianNetwork([('Exercise', 'HeartDisease'), ('Diet', 'HeartDisease')])

# Define CPDs
cpd_exercise = TabularCPD('Exercise', 2, [[0.7], [0.3]])
cpd_diet = TabularCPD('Diet', 2, [[0.6], [0.4]])
cpd_hd = TabularCPD('HeartDisease', 2,
                    [[0.9, 0.8, 0.7, 0.1],
                     [0.1, 0.2, 0.3, 0.9]],
                    evidence=['Exercise', 'Diet'],
                    evidence_card=[2, 2])

model.add_cpds(cpd_exercise, cpd_diet, cpd_hd)
model.check_model()

inference = VariableElimination(model)
result = inference.query(variables=['HeartDisease'], evidence={'Exercise': 1,
'Diet': 0})
print(result)
```

---

9. Write a python program to implement K-Nearest Neighbor.

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

```
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target)

model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)

print("Accuracy:", model.score(X_test, y_test))
```

10. Write a python program to implement linear regression.

```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

X, y = make_regression(n_samples=100, n_features=1, noise=10)
X_train, X_test, y_train, y_test = train_test_split(X, y)

model = LinearRegression()
model.fit(X_train, y_train)

print("Coefficient:", model.coef_)
print("Intercept:", model.intercept_)
print("Score:", model.score(X_test, y_test))
```

11. Write a python program to implement multiple regression.

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import numpy as np

# Sample data: features [hours studied, attendance percentage]
X = np.array([[5, 80], [10, 90], [2, 60], [8, 85], [7, 75]])
y = np.array([60, 85, 45, 80, 70])

X_train, X_test, y_train, y_test = train_test_split(X, y)

model = LinearRegression()
model.fit(X_train, y_train)

print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
print("Score:", model.score(X_test, y_test))
```

12. Write a python program to implement decision trees and random forests.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
print("Decision Tree Accuracy:", dt.score(X_test, y_test))

# Random Forest
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)
print("Random Forest Accuracy:", rf.score(X_test, y_test))
```

---

13. Write a program to build SVM models for classification and regression using scikit-learn.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC, SVR

# Classification
iris = datasets.load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target)
clf = SVC()
clf.fit(X_train, y_train)
print("SVM Classification Accuracy:", clf.score(X_test, y_test))

# Regression
X, y = datasets.make_regression(n_samples=100, n_features=1, noise=10)
X_train, X_test, y_train, y_test = train_test_split(X, y)
reg = SVR()
reg.fit(X_train, y_train)
print("SVM Regression Score:", reg.score(X_test, y_test))
```

---

14. Write a program to implement Random Forest and Gradient Boosting using scikit-learn.

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import train_test_split

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Random Forest
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
print("Random Forest Accuracy:", rf.score(X_test, y_test))

# Gradient Boosting
gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)
print("Gradient Boosting Accuracy:", gb.score(X_test, y_test))
```

15. Write a python program to implement K-means using scikit-learn.

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

X, _ = make_blobs(n_samples=100, centers=3, n_features=2)

kmeans = KMeans(n_clusters=3)
kmeans.fit(X)

plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
color='red')
plt.title("K-Means Clustering")
plt.show()
```

16. Write a python program to implement Agglomerative Hierarchical Clustering using scikit-learn.

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

X, _ = make_blobs(n_samples=100, centers=3)

cluster = AgglomerativeClustering(n_clusters=3)
labels = cluster.fit_predict(X)

plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.title("Agglomerative Clustering")
plt.show()
```

17. Write a python program to implement the EM algorithm for Bayesian networks in Python.

```
from pgmpy.models import BayesianNetwork
from pgmpy.inference import BayesianModelSampling
from pgmpy.estimators import ExpectationMaximization as EM
from pgmpy.estimators import BayesianEstimator
from pgmpy.factors.discrete import TabularCPD

# Define a simple model structure
model = BayesianNetwork([('A', 'C'), ('B', 'C')])

# Simulate data
cpd_a = TabularCPD('A', 2, [[0.6], [0.4]])
cpd_b = TabularCPD('B', 2, [[0.7], [0.3]])
cpd_c = TabularCPD('C', 2,
                   [[0.9, 0.6, 0.7, 0.1],
                    [0.1, 0.4, 0.3, 0.9]],
                   evidence=['A', 'B'], evidence_card=[2, 2])

model.add_cpds(cpd_a, cpd_b, cpd_c)

sampler = BayesianModelSampling(model)
data = sampler.forward_sample(size=1000)

# Learn with EM
model.fit(data, estimator=EM, prior_type='BDeu', complete_samples_only=False)
print("EM fitting completed.")
```

---

18. Write a python program to build a simple neural network using Keras.

```
from keras.models import Sequential
from keras.layers import Dense
import numpy as np

X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])

model = Sequential()
model.add(Dense(5, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=100, verbose=0)

print("Predictions:", model.predict(X))
```

---

19. Write a python program to build a deep neural network (MLP) using Keras.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

X, y = load_iris(return_X_y=True)
y = to_categorical(y)

X_train, X_test, y_train, y_test = train_test_split(X, y)

model = Sequential()
model.add(Dense(64, input_dim=4, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(3, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(X_train, y_train, epochs=100, verbose=0)

loss, accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", accuracy)
```

20. Write a python program for implementation of Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms.

```
# BFS
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])
    while queue:
        node = queue.popleft()
        if node not in visited:
            print(node, end=" ")
            visited.add(node)
            queue.extend(graph[node])

# DFS
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    print(start, end=" ")
    visited.add(start)
    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)

# Example usage
graph = {
    'A': ['B', 'C'], 'B': ['D', 'E'],
    'C': ['F'], 'D': [], 'E': [], 'F': []
}

print("BFS:")
bfs(graph, 'A')
print("\nDFS:")
dfs(graph, 'A')
```

Batch 2 :

1. Develop a code by implementing the Uninformed search algorithm - BFS.

```
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])

    while queue:
        node = queue.popleft()
        if node not in visited:
            print(node, end=" ")
            visited.add(node)
            queue.extend(neighbor for neighbor in graph[node] if neighbor not in
visited)

graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [], 'E': [], 'F': []
}

bfs(graph, 'A')
```

---

2. Develop a code by implementing the Uninformed search algorithm - DFS.

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    if start not in visited:
        print(start, end=" ")
        visited.add(start)
        for neighbor in graph[start]:
            dfs(graph, neighbor, visited)

graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [], 'E': [], 'F': []
}

dfs(graph, 'A')
```

---

3. Develop a code by implementing the Informed search algorithm - A.*

```
from queue import PriorityQueue

def a_star(graph, start, goal, heuristic):
    open_set = PriorityQueue()
    open_set.put((0, start))
    came_from = {}
    g_score = {node: float('inf') for node in graph}
    g_score[start] = 0

    while not open_set.empty():
        current = open_set.get()[1]
        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]

        for neighbor, cost in graph[current]:
            temp_g_score = g_score[current] + cost
            if temp_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = temp_g_score
                f_score = temp_g_score + heuristic[neighbor]
                open_set.put((f_score, neighbor))

    return []

graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('D', 4), ('E', 2)],
    'C': [('F', 5)],
    'D': [], 'E': [], 'F': []
}
heuristic = {'A': 6, 'B': 4, 'C': 4, 'D': 0, 'E': 0, 'F': 0}

print("A* path:", a_star(graph, 'A', 'D', heuristic))
```

---

4. Develop a code by implementing the Informed search algorithm - Memory Bounded A.*

Simplified version of **Simplified MA** (SMA*):

```
# Simplified Memory Bounded A* using BFS + Heuristic
from queue import PriorityQueue

def memory_bounded_a_star(graph, start, goal, heuristic, memory_limit=3):
    open_list = PriorityQueue()
    open_list.put((heuristic[start], start))
    visited = set()

    while not open_list.empty():
        if open_list.qsize() > memory_limit:
            open_list.get()
        f, current = open_list.get()
        visited.add(current)

        if current == goal:
            return f"Goal {goal} reached!"

        for neighbor, cost in graph.get(current, []):
            if neighbor not in visited:
                open_list.put((heuristic[neighbor], neighbor))

    return "Goal not found"

graph = {
    'A': [('B', 1), ('C', 3)],  'B': [('D', 4)],
    'C': [('E', 2)], 'D': [],  'E': []
}
heuristic = {'A': 5, 'B': 3, 'C': 2, 'D': 0, 'E': 0}

print(memory_bounded_a_star(graph, 'A', 'D', heuristic))
```

5. Develop a code by implementing the Analysis/action of data set using naive Bayes models.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

data = load_iris()
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target,
test_size=0.3)

model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
```

6. Develop a code by implementing the probability relationship check between two dataset using Bayesian Networks.

```
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination

# Define model structure
model = BayesianNetwork([('Exercise', 'HeartDisease'), ('Diet', 'HeartDisease')])

# Define CPDs
cpd_exercise = TabularCPD(variable='Exercise', variable_card=2, values=[[0.6],
[0.4]])
cpd_diet = TabularCPD(variable='Diet', variable_card=2, values=[[0.7], [0.3]])
cpd_hd = TabularCPD(
    variable='HeartDisease', variable_card=2,
    values=[[0.9, 0.6, 0.7, 0.1],
            [0.1, 0.4, 0.3, 0.9]],
    evidence=['Exercise', 'Diet'],
    evidence_card=[2, 2]
)

# Add CPDs to the model
model.add_cpds(cpd_exercise, cpd_diet, cpd_hd)

# Check model validity
assert model.check_model()

# Perform inference
infer = VariableElimination(model)
result = infer.query(variables=['HeartDisease'], evidence={'Exercise': 1, 'Diet':
0})
print(result)
```

7. Develop a code to understand and predict an outcome variable based on the input Regression models.

(Linear Regression example)

```
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Sample data
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([2, 4, 5, 4, 5])

model = LinearRegression()
model.fit(X, y)

# Prediction
y_pred = model.predict(X)

print("Predictions:", y_pred)

# Plot
plt.scatter(X, y, color='blue')
plt.plot(X, y_pred, color='red')
plt.title('Linear Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.show()
```

---

8. Develop a code to Build a decision tree to predict the expected output from the desired input.

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data = load_iris()
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target,
test_size=0.3)

model = DecisionTreeClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
```

---

9. Develop a code to build random forests for the dataset by understanding the difference between Random Forest and Decision Tree.

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)
```

```
y_pred_rf = model.predict(X_test)

print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
```

Note: Compared to a single Decision Tree, Random Forest improves accuracy by reducing overfitting using multiple trees.

---

10. Develop a code to Build Support Vector Machine models for Classification Task.

```
from sklearn.svm import SVC

model = SVC(kernel='linear')  # or use 'rbf'
model.fit(X_train, y_train)
y_pred_svm = model.predict(X_test)

print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
```

---

11. Write a program to build Support Vector Machine models for classification task.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report

# Load dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Create SVM model
model = SVC(kernel='linear')
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)
print("Classification Report:\n", classification_report(y_test, y_pred))
```

---

12. Implement an application that imposes the ensembling techniques.

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

iris = load_iris()
X = iris.data
```

```
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Random Forest
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, rf_pred))

# Gradient Boosting
gb = GradientBoostingClassifier(n_estimators=100)
gb.fit(X_train, y_train)
gb_pred = gb.predict(X_test)
print("Gradient Boosting Accuracy:", accuracy_score(y_test, gb_pred))
```

---

13. Implement an application that uses clustering algorithms.

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

# Generate sample data
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60)

# Apply KMeans clustering
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

# Visualize
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=200,
c='red', marker='X')
plt.title("KMeans Clustering")
plt.show()
```

---

14. Write EM code for implementing Bayesian Networks.

```
from pgmpy.models import BayesianNetwork
from pgmpy.inference import VariableElimination
from pgmpy.estimators import ExpectationMaximization, BayesianEstimator
from pgmpy.estimators import ParameterEstimator
from pgmpy.factors.discrete import TabularCPD
import pandas as pd

# Create dummy data
data = pd.DataFrame(data={'Rain': [1, 0, 1, 0, 1],
                          'Traffic': [1, 1, 0, 0, 1],
                          'Late': [1, 1, 0, 0, 1]})

model = BayesianNetwork([('Rain', 'Traffic'), ('Traffic', 'Late')])
model.fit(data, estimator=BayesianEstimator)

inference = VariableElimination(model)
```

```
print("P(Late=1 | Rain=1):")
print(inference.query(variables=['Late'], evidence={'Rain': 1}))
```

---

15. Develop a deep-learning Neural Network for experiencing different architectural model.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical

digits = load_digits()
X = digits.data / 16.0
y = to_categorical(digits.target)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = Sequential()
model.add(Dense(128, input_shape=(64,), activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test))
```

---

16. Write a code to find the shortest path between two points by evaluating the cost function of each possible path. (Using A* Algorithm)

```
import heapq

def a_star(graph, start, goal, h):
    queue = [(0 + h[start], 0, start, [])]
    visited = set()

    while queue:
        (f, cost, node, path) = heapq.heappop(queue)
        if node in visited:
            continue
        path = path + [node]
        if node == goal:
            return path
        visited.add(node)

        for neighbor, weight in graph[node]:
            if neighbor not in visited:
                heapq.heappush(queue, (cost + weight + h[neighbor], cost + weight,
neighbor, path))

    return None

graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('D', 3)],
    'C': [('D', 1)],  'D': []
}

heuristic = {'A': 4, 'B': 2, 'C': 2, 'D': 0}
print(a_star(graph, 'A', 'D', heuristic))
```

---

17. Develop a simple task that uses a limited amount of memory and is suitable for large search spaces. (Memory-Bounded A*)

```
def memory_bounded_a_star(start, goal, h, successors):
    frontier = [(h[start], start, [start])]
    limit = 10  # memory constraint

    while frontier:
        frontier.sort()
        if len(frontier) > limit:
            frontier.pop()

        (f, node, path) = frontier.pop(0)
        if node == goal:
            return path

        for (child, cost) in successors[node]:
            new_path = path + [child]
            new_cost = len(new_path) + h[child]
            frontier.append((new_cost, child, new_path))

    return None

successors = {
    'S': [('A', 1), ('B', 4)],
    'A': [('C', 1)],  'B': [('C', 2)],
    'C': [('G', 3)],  'G': []
}
h = {'S': 7, 'A': 6, 'B': 5, 'C': 3, 'G': 0}
print(memory_bounded_a_star('S', 'G', h, successors))
```

---

18. Develop a Quiz application using conditional probability.

```
# Simple simulation using Bayes' Theorem

def bayes_theorem(p_a, p_b_given_a, p_b):
    return (p_b_given_a * p_a) / p_b

# Example: Probability of passing given studied
p_study = 0.7
p_pass_given_study = 0.9
p_pass = 0.6

p_study_given_pass = bayes_theorem(p_study, p_pass_given_study, p_pass)
print("P(Studied | Passed):", round(p_study_given_pass, 2))
```

19. Application showing basic conditional dependencies (Bayesian Network Skeleton)

```
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD

model = BayesianNetwork([('Studied', 'Passed'), ('AttendedClass', 'Passed')])

cpd_studied = TabularCPD('Studied', 2, [[0.3], [0.7]])
cpd_attended = TabularCPD('AttendedClass', 2, [[0.4], [0.6]])
cpd_passed = TabularCPD('Passed', 2,
                        [[0.95, 0.8, 0.7, 0.1], [0.05, 0.2, 0.3, 0.9]],
                        evidence=['Studied', 'AttendedClass'],
                        evidence_card=[2, 2])

model.add_cpds(cpd_studied, cpd_attended, cpd_passed)
print("Bayesian Network created with dependencies: Studied -> Passed, AttendedClass
-> Passed")
```

20. Develop a regression model that predicts a continuous numerical outcome.

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_regression
import matplotlib.pyplot as plt

X, y = make_regression(n_samples=100, n_features=1, noise=15)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = LinearRegression()
model.fit(X_train, y_train)

print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
print("Score:", model.score(X_test, y_test))

# Plot
plt.scatter(X, y, color='blue')
plt.plot(X, model.predict(X), color='red')
plt.title("Linear Regression")
plt.xlabel("X")
plt.ylabel("y")
plt.show()
```