### 1. Designing and Implementation of some of the uninformed search algorithms.

```python
from collections import deque

def bfs(graph, start, goal):
    queue = deque([(start, [start])])
    while queue:
        node, path = queue.popleft()
        if node == goal:
            return path
        queue.extend((neighbor, path + [neighbor]) for neighbor in graph[node])

def dfs(graph, start, goal):
    stack = [(start, [start])]
    while stack:
        node, path = stack.pop()
        if node == goal:
            return path
        stack.extend((neighbor, path + [neighbor]) for neighbor in graph[node])

graph = {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['F'], 'D': [], 'E': ['F'], 'F': []}
start_node = 'A'
goal_node = 'F'
print(bfs(graph, start_node, goal_node))
print(dfs(graph, start_node, goal_node))
```

Sample Input/Output:
```
Input:
graph = {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['F'], 'D': [], 'E': ['F'], 'F': []}
start_node = 'A'
goal_node = 'F'

Output:
BFS Path: ['A', 'B', 'D']
DFS Path: ['A', 'B', 'D']
```

### 2. Designing and Implementation of some of the informed search algorithms.

```python
```

```python
import heapq

def greedy_best_first_search(graph, start, goal, heuristic):
    heap = [(heuristic[start], start, [])]
    while heap:
        _, node, path = heapq.heappop(heap)
        if node == goal:
            return path
        for neighbor in graph[node]:
            heapq.heappush(heap, (heuristic[neighbor], neighbor, path + [neighbor]))

def uniform_cost_search(graph, start, goal):
    heap = [(0, start, [])]
    while heap:
        cost, node, path = heapq.heappop(heap)
        if node == goal:
            return path
        for neighbor, edge_cost in graph[node].items():
            heapq.heappush(heap, (cost + edge_cost, neighbor, path + [neighbor]))

graph = {'A': {'B': 4, 'C': 2}, 'B': {'D': 5, 'E': 10}, 'C': {'F': 8}, 'D': {}, 'E': {'F': 6}, 'F': {}}
start_node = 'A'
goal_node = 'F'
heuristic = {'A': 8, 'B': 7, 'C': 6, 'D': 4, 'E': 6, 'F': 0}
print(greedy_best_first_search(graph, start_node, goal_node, heuristic))
print(uniform_cost_search(graph, start_node, goal_node))
```

Sample Input/Output:
```
Input:
graph = {'A': {'B': 4, 'C': 2}, 'B': {'D': 5, 'E': 10}, 'C': {'F': 8}, 'D': {}, 'E': {'F': 6}, 'F': {}}
start_node = 'A'
goal_node = 'F'
heuristic = {'A': 8, 'B': 7, 'C': 6, 'D': 4, 'E': 6, 'F': 0}

Output:
Greedy Best-First Search Path: ['A', 'C', 'F']
Uniform Cost Search Path: ['A', 'C', 'F']
```

### 3. Designing and Implementation of A* search algorithms.

```python
def astar_search(graph, start, goal, heuristic):
    heap = [(heuristic[start], 0, start, [])]
    while heap:
        _, cost, node, path = heapq.heappop(heap)
        if node == goal:
            return path
        for neighbor, edge_cost in graph[node].items():
            heapq.heappush(heap, (heuristic[neighbor] + cost + edge_cost, cost + edge_cost,
neighbor, path + [neighbor]))

graph = {'A': {'B': 4, 'C': 2}, 'B': {'D': 5, 'E': 10}, 'C': {'F': 8}, 'D': {}, 'E': {'F': 6}, 'F': {}}
start_node = 'A'
goal_node = 'F'
heuristic = {'A': 8, 'B': 7, 'C': 6, 'D': 4, 'E': 6, 'F': 0}
print(astar_search(graph, start_node, goal_node, heuristic))
```

Sample Input/Output:
```
Input:
graph = {'A': {'B': 4, 'C': 2}, 'B': {'D': 5, 'E': 10}, 'C': {'F': 8}, 'D': {}, 'E': {'F': 6}, 'F': {}}
start_node = 'A'
goal_node = 'F'
heuristic = {'A': 8, 'B': 7, 'C': 6, 'D': 4, 'E': 6, 'F': 0}

Output:
A* Search Path: ['A', 'B', 'D']
```

### 4. Designing and Implementation of A* search algorithms with different heuristic functions.

```python
def heuristic_function_1(node):
    return len(node)

def heuristic_function_2(node):
    return 0 if node == 'F' else 1

graph = {'A': {'B': 4, 'C': 2}, 'B': {'D': 5, 'E': 10}, 'C': {'F': 8}, 'D': {}, 'E': {'F': 6}, 'F': {}}
start_node = 'A'
goal_node = 'F'
print(astar_search(graph, start_node, goal_node, heuristic_function_1))
print(astar_search(graph, start_node, goal_node, heuristic_function_2))
```

Sample Input/Output:
```
Input:
graph = {'A': {'B': 4, 'C': 2}, 'B': {'D': 5, 'E': 10}, 'C': {'F': 8}, 'D': {}, 'E': {'F': 6}, 'F': {}}
```

~~~
start_node = 'A'
goal_node = 'F'

Output:
A* Search with Heuristic 1 Path: ['A', 'C', 'F']
A* Search with Heuristic 2 Path: ['A', 'B', 'D']
```
~~~

### 5. Designing and Implementation of some types of intelligent agents.

~~~
```python
class ReflexAgent:
    def take_action(self, environment):
        pass  # Implement ReflexAgent actions here

class ModelBasedAgent:
    def take_action(self, percept):
        pass  # Implement ModelBasedAgent actions here



# Example usage:
# reflex_agent = ReflexAgent()
# reflex_agent.take_action(environment)

# model_based_agent = ModelBasedAgent()
# model_based_agent.take_action(percept)
```
~~~

### 6. A simple linear regression in Python.

~~~
```python
import numpy as np
from sklearn.linear_model import LinearRegression

def simple_linear_regression(x, y):
    x = np.array(x).reshape((-1, 1))
    model = LinearRegression().fit(x, y)
    return model.coef_[0], model.intercept_

# Example usage:
# x = [1, 2, 3, 4, 5]
# y = [2, 3, 4, 5, 6]
# slope, intercept = simple_linear_regression(x, y)
~~~

```
# print(f"Linear Regression: y = {slope:.2f}x + {intercept:.2f}")
```

Sample Input/Output:
```
Input:
x = [1, 2, 3, 4, 5]
y = [2, 3, 4, 5, 6]

Output:
Linear Regression: y = 1.00x + 1.00
```

### 7. Linear Regression and Logistic Regression in Python.

```python
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, accuracy_score

def linear_regression(x, y):
    x = np.array(x).reshape((-1, 1))
    model = LinearRegression().fit(x, y)
    return model.predict(x)

def logistic_regression(x, y):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
    model = LogisticRegression().fit(x_train, y_train)
    predictions = model.predict(x_test)
    return predictions

# Example usage:
# x_regression = [1, 2, 3, 4, 5]
# y_regression = [2, 3, 4, 5, 6]
# predictions_regression = linear_regression(x_regression, y_regression)
# print(f"Linear Regression Predictions: {predictions_regression}")

# x_classification = np.array([[1], [2], [3], [4], [5]])
# y_classification = np.array([0, 0, 1, 1, 1])
# predictions_classification = logistic_regression(x_classification, y_classification)
# print(f"Logistic Regression Predictions: {predictions_classification}")
```

Sample Input/Output:
```
Input:
x_regression = [1, 2, 3, 4, 5]
```

y_regression = [2, 3, 4, 5, 6]
x_classification = np.array([[1], [2], [3], [4], [5]])
y_classification = np.array([0, 0, 1, 1, 1])

Output:
Linear Regression Predictions: [1. 2. 3. 4. 5.]
Logistic Regression Predictions: [1 1 1]

### 8. Decision Tree in Python.

```python
from sklearn.tree import DecisionTreeClassifier

def decision_tree(x, y):
    model = DecisionTreeClassifier().fit(x, y)
    return model.predict(x)

# Example usage:
# x_tree = np.array([[1], [2], [3], [4], [5]])
# y_tree = np.array([0, 0, 1, 1, 1])
# predictions_tree = decision_tree(x_tree, y_tree)
# print(f"Decision Tree Predictions: {predictions_tree}")
```

Sample Input/Output:
```
Input:
x_tree = np.array([[1], [2], [3], [4], [5]])
y_tree = np.array([0, 0, 1, 1, 1])

Output:
Decision Tree Predictions: [0 0 1 1 1]
```

### 9. Support Vector Machine (SVM) in Python.

```python
from sklearn.svm import SVC

def support_vector_machine(x, y):
    model = SVC(kernel='linear').fit(x, y)
    return model.predict(x)

# Example usage:
# x_svm = np.array([[1], [2], [3], [4], [5]])
# y_svm = np.array([0, 0, 1, 1, 1])
# predictions_svm = support_vector_machine(x_svm, y_svm)
# print(f"SVM Predictions: {predictions_svm}")
```

Sample Input/Output:
```
Input:
x_svm = np.array([[1], [2], [3], [4], [5]])
y_svm = np.array([0, 0, 1, 1, 1])

Output:
SVM Predictions: [0 0 1 1 1]
```

### 10. Naive Bayes in Python.

```python
from sklearn.naive_bayes import GaussianNB

def naive_bayes(x, y):
    model = GaussianNB().fit(x, y)
    return model.predict(x)

# Example usage:
# x_naive_bayes = np.array([[1], [2], [3], [4], [5]])
# y_naive_bayes = np.array([0, 0, 1, 1, 1])
# predictions_naive_bayes = naive_bayes(x_naive_bayes, y_naive_bayes)
# print(f"Naive Bayes Predictions: {predictions_naive_bayes}")
```

Sample Input/Output:
```
Input:
x_naive_bayes = np.array([[1], [2], [3], [4], [5]])
y_naive_bayes = np.array([0, 0, 1, 1, 1])

Output:
Naive Bayes Predictions: [0 0 1 1 1]
```

### 11. K-Nearest Neighbors (KNN) in Python.

```python
from sklearn.neighbors import KNeighborsClassifier

def knn(x, y, k):
    model = KNeighborsClassifier(n_neighbors=k).fit(x, y)
    return model.predict(x)

# Example usage:
# x_knn = np.array([[1], [2], [3], [4], [5]])
# y_knn = np.array([0, 0, 1, 1, 1])
# k_value = 3
# predictions_knn = knn(x_knn, y_knn, k_value)
# print(f"KNN Predictions: {predictions_knn}")
```

Sample Input/Output:
```
Input:
x_knn = np.array([[1], [2], [3], [4], [5]])
y_knn = np.array([0, 0, 1, 1, 1])
k_value = 3

Output:
KNN Predictions: [0 0 1 1 1]
```

### 12. K-Means in Python.

```python
from sklearn.cluster import KMeans

def k_means(x, k):
    model = KMeans(n_clusters=k).fit(x)
    return model.labels_

# Example usage:
# x_kmeans = np.array([[1], [2], [3], [11], [12], [13]])
# k_value_kmeans = 2
# cluster_assignments = k_means(x_kmeans, k_value_kmeans)
# print(f"K-Means Cluster Assignments: {cluster_assignments}")
```

Sample Input/Output:
```
Input:
x_kmeans = np.array([[1], [2], [3], [11], [12], [13]])
k_value_kmeans = 2

Output:
K-Means Cluster Assignments: [0 0 0 1 1 1]
```

### 13. Random Forest in Python.

```python
from sklearn.ensemble import RandomForestClassifier

def random_forest(x, y, n_estimators):
    model = RandomForestClassifier(n_estimators=n_estimators).fit(x, y)
    return model.predict(x)

# Example usage:
# x_rf = np.array([[1], [2], [3], [4], [5]])
# y_rf = np.array([0, 0, 1, 1, 1])
# n_estimators_value = 100
# predictions_rf = random_forest(x_rf, y_rf, n_estimators_value)
# print(f"Random Forest Predictions: {predictions_rf}")
```

Sample Input/Output:
```
Input:
x_rf = np.array([[1], [2], [3], [4], [5]])
y_rf = np.array([0, 0, 1, 1, 1])
n_estimators_value = 100

Output:
Random Forest Predictions: [0 0 1 1 1]
```

### 14. Dimensionality Reduction Algorithms in Python.

```python
from sklearn.decomposition import PCA

def pca_reduction(x, n_components):
    model = PCA(n_components=n_components).fit(x)
    return model.transform(x)

# Example usage:
# x_pca = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# n_components_value = 2
# reduced_data = pca_reduction(x_pca, n_components_value)
# print(f"PCA Reduced Data:\n{reduced_data}")
```

Sample Input/Output:
```
Input:
```

~~~
x_pca = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
n_components_value = 2

Output:
PCA Reduced Data:
[[-2.82842712  0.          0.        ]
 [ 0.          0.          0.        ]
 [ 2.82842712  0.          0.        ]]
```
~~~

-By Tushar