Faculty of Computer & Information Sciences
Ain Shams University
Subject: CSC 343 Artificial Intelligence
Year: (3rd year)undergraduate
Academic year: 2nd term 2019-2020

Research Topic (1)

Title: Classification, Neural Computing and Search

تحذير هام: علي الطالب عدم كتابة اسمه أو كتابة اي شيء يدل علي شخصيته

1. Introduction

1.1 BFS:

➢ **When:**

- use BFS - when you want to find the shortest path from a certain source node to a certain destination.
- The smallest number of steps to reach the end state from a given initial state.
- For unweighted graph to find shortest path and minimal steps.
- Not need to traverse all nodes.

➢ **Apps:**

- Find Connected Components in Undirected Graph
- Web Crawling
- Social N/W: Friend Finder
- Pocket Cube (2x2x2 Rubic Cube)
- Garbage Collector.

1.2 Neural network:

➢ **When:**

- simulate the learning process and generalization ability of the human brain.
- Classification of data.
- It is good time to start investing in stock market.

➢ **Apps:**

- Speech recognition.
- Audio generation.
- Time series analysis.
- auto-piloting in Space.

1.3 ID3:

- **When:**
  - when reduce Cost sensitive Decision Tree.
  - Predicting Heater Outlet Temperature.
- **Apps:**
  - on Food Database.
  - Identifying Cancer.

2. The algorithms

2.1. Breadth-first search

2.1.1 The main steps of the algorithm.

1. Find the start node.
2. Make it visited and put it in queue.
3. Loop still the queue not empty.
4. Hold the front of the queue and visit all its Neighbors that not visited.
5. Push the valid neighbors in queue.
6. Repeat step 4, 5 until the end.

2.1.2 The implementation of the algorithm (your Python code)

```python
# region SearchAlgorithms
class Positions:
    def __init__(self, r, c):
        self.row = r
        self.col = c

class Node:
    id = None
    up = None
    down = None
    left = None
    right = None
    previousNode = None
    position_node = None

    def __init__(self, value):
        self.value = value
```

```python
class SearchAlgorithms:
    path = []  # Represents the correct path 
    fullPath = []  # Represents all visited n

    maze = []
    number_of_col = 0
    number_of_row = 0

    start_point_row = 0
    start_point_col = 0
              #U  D  L   R
    rowNum = [-1, 1, 0, 0]
    colNum = [0, 0, -1, 1]
    def __init__(self, mazeStr):
        ''' mazeStr contains the full board T
        self.mazeStr = mazeStr
        self.length = len(mazeStr)
```

```python
    def Create_maze(self):
        row = []
        id = 0
        for i in range(self.length):
            if self.mazeStr[i] != " " and self.mazeStr[i] != ",":
                node = Node(self.mazeStr[i])
                node.id = id
                id+=1
                row.append(node)
            elif self.mazeStr[i] == " ":
                self.maze.append(row)
                row = []
        self.maze.append(row)

        self.number_of_col = self.maze[0]
        self.number_of_row = self.maze

        self.maze[0][0].position_node = Positions(0, 0)
```

```python
        self.maze[0][0].position_node = Positions(0, 0)
        for i in range(len(self.maze)):
            for j in range(len(self.maze[i])):
                #print(self.maze[i][j].value, end=" ")
                self.maze[i][j].position_node = Positions(i, j)

                for d in range(4):
                    row = self.maze[i][j].position_node.row + self.rowNum[d]
                    col = self.maze[i][j].position_node.col + self.colNum[d]

                    if self.Valid_direction(row, col):
                        if d == 0:
                            self.maze[i][j].up = self.maze[row][col]
                        elif d == 2:
                            self.maze[i][j].left = self.maze[row][col]
                        elif d == 3:
                            self.maze[i][j].right = self.maze[row][col]
                        elif d == 1:
                            self.maze[i][j].down = self.maze[row][col]
```

```python
def Valid_direction(self, i, j):
    if not (0 <= i and i < len(self.maze) and 0 <= j and j < len(self.maze[0])):
        return False
    return True

def Find_Start(self):
    for i in range(len(self.maze)):
        for j in range(len(self.maze[i])):
            if self.maze[i][j].value == "S":
                self.start_point_row = i
                self.start_point_col = j

def Find_end(self, i, j):
    if self.maze[i][j].value == "E":
        print("Found")
        return True
    return False
```

```python
def Valid_move(self, i, j, visited, node):
    if not (0 <= i and i < len(self.maze) and 0 <= j and j < len(self.maze[0])):
        return False
    elif self.maze[i][j].value == "#":
        return False
    elif visited[i][j]:
        return False
    elif node == None:
        return False
    return True
```

```python
def BFS(self):
    '''Implement Here'''
    self.Create_maze()
    self.Find_Start()

    queue_direction = deque()

    visited = [[False for i in range(len(self.number_of_col))] for j in range(len(self.number_of_row))]

    visited[self.start_point_row][self.start_point_col] = True
    self.maze[self.start_point_row][self.start_point_col].position_node = Positions(self.start_point_row,
                                                                                   self.start_point_col)
    self.maze[self.start_point_row][self.start_point_col].previousNode = Positions(-1, -1)

    queue_direction.append(self.maze[self.start_point_row][self.start_point_col])
```

```python
    while queue_direction:
        currend_node = queue_direction.popleft()

        if(self.Find_end(currend_node.position_node.row, currend_node.position_node.col)):
            self.fullPath.append(currend_node.id)
            prev_position = Positions(0, 0)
            previous_node = currend_node
            while previous_node.value != 'S':
                r = previous_node.position_node.row
                c = previous_node.position_node.col
                self.path.append(self.maze[r][c].id)
                previous_node = self.maze[r][c].previousNode
            break
        cur_pos_row = currend_node.position_node.row
        cur_pos_col = currend_node.position_node.col

        if self.Valid_move(cur_pos_row - 1, cur_pos_col, visited, currend_node.up) :
            self.maze[cur_pos_row - 1][cur_pos_col].previousNode = currend_node
            visited[cur_pos_row- 1][cur_pos_col] = True
```

```python
if self.Valid_move(cur_pos_row - 1, cur_pos_col, visited, currend_node.up) :
    self.maze[cur_pos_row - 1][cur_pos_col].previousNode = currend_node
    visited[cur_pos_row- 1][cur_pos_col] = True
    queue_direction.append(self.maze[cur_pos_row- 1][cur_pos_col])

if self.Valid_move(cur_pos_row + 1, cur_pos_col, visited, currend_node.down):
    self.maze[cur_pos_row + 1][cur_pos_col].previousNode = currend_node
    visited[cur_pos_row + 1][cur_pos_col] = True
    queue_direction.append(self.maze[cur_pos_row + 1][cur_pos_col])

if self.Valid_move(cur_pos_row, cur_pos_col - 1, visited, currend_node.left):
    self.maze[cur_pos_row][cur_pos_col - 1].previousNode = currend_node
    visited[cur_pos_row][cur_pos_col - 1] = True
    queue_direction.append(self.maze[cur_pos_row][cur_pos_col - 1])
if self.Valid_move(cur_pos_row, cur_pos_col + 1, visited, currend_node.right):
    self.maze[cur_pos_row][cur_pos_col + 1].previousNode = currend_node
    visited[cur_pos_row][cur_pos_col + 1] = True
    queue_direction.append(self.maze[cur_pos_row][cur_pos_col + 1])
```

```python
            '''...'''
            self.fullPath.append(currend_node.id)
        self.path.reverse()
        return self.fullPath, self.path

# endregion
```

2.1.3 Sample run (the output)

```
**BFS**
 Full Path is: [0, 7, 1, 14, 2, 21, 9, 22, 16, 10, 29, 17, 11, 18, 4, 25, 19, 5, 32, 26, 20, 6, 31]
 Path: [1, 2, 9, 16, 17, 18, 25, 32, 31]
```

2.2 Neural network

2.2.1 The main steps of the algorithm

1. Initialize the weights.
2. Update weights to correct data by expected output.
3. Get the summation of weights multiply the value of input.
4. Add bias to input.
5. Check the output of each step to get correct data.

2.2.2 The implementation of the algorithm (your Python code)

```
# region NeuralNetwork
class NeuralNetwork():

    def __init__(self, learning_rate, threshold):
        self.learning_rate = learning_rate
        self.threshold = threshold
        np.random.seed(1)
        self.synaptic_weights = 2* np.random.random((2, 1)) - 1

    def step(self, x):
        if x > float(self.threshold):
            return 1
        else:
            return 0
```

```
    def learn (self, inputs):
        weights = self.synaptic_weights.tolist()
        # print(weights[1][0])
        inputs = inputs.astype(float)
        temp_w = -5
        w = 0
        for i in inputs:
            c = 0
            for j in i:
                if c == 2:
                    w += j * temp_w
                    continue
                w += j * weights[c][0]
                c += 1
        output = self.step(w)
        return output
```

```python
def train(self, training_inputs, training_outputs, training_iterations):
    """..."""
    basis = [1]
    temp = []
    for row in training_inputs:
        modified_row = np.append(row, basis)
        temp.append(modified_row)
    temp = np.array(temp)
    training_inputs_basis = temp

    for iteration in range(training_iterations):
        output = self.learn(training_inputs_basis)
        error = training_outputs - output
        adjustments = np.dot(training_inputs_basis.T, error * self.learning_rate)
        temp_adjustments = []
        temp_adjustments.append(adjustments[0])
        temp_adjustments.append(adjustments[1])
        self.synaptic_weights += temp_adjustments
```

```python
    def think(self, inputs):
        weights = self.synaptic_weights.tolist()
        inputs = inputs.astype(float)
        temp_w = -5
        w = 0
        c = 0
        for i in range(len(inputs)+1):
            if c == 2:
                w += 2 * temp_w
                continue
            w += inputs[i] * weights[c][0]
            c += 1
        #print("w = ", w)
        output = self.step(w)
        return output
# endregion
```

2.2.3 Sample run (the output)

```
Beginning Randomly Generated Weights:
[[-0.16595599]
 [ 0.44064899]]
Ending Weights After Training:
[[4.63404401]
 [5.24064899]]
Considering New Situation:  1 1 New Output data:  1
```

2.3 ID3

2.3.1 The main steps of the algorithm

1. Calculate the entropy of every attribute using the data set.
2. Split the set into subsets using the attribute for which information gain is the maximum
3. Make a decision tree node containing that attribute
4. Recurse on subsets using remaining attributes

2.3.2 The implementation of the algorithm (your Python code)

```
class Node_Feature:
    name = None
    left = None
    right = None
class ID3:
    Total_entropy = 0
    max_name = ""
    max_gain = -1
    col_age         = []
    col_prescription= []
    col_astigmatic  = []
    col_tearRate    = []
    col_diabetic    = []
    col_needLense   = []
    #
    dictionary_nodes = {}
    rec_col_zero_age = []
    rec_col_one_age = []
```

```
def __init__(self, features):
    self.features = features

def set_all_data(self):
    trained_data = item.getDataset()
    for i in range(len(trained_data)):
        self.col_age.append(trained_data[i].age)
        self.col_prescription.append(trained_data[i].prescription)
        self.col_astigmatic.append(trained_data[i].astigmatic)
        self.col_tearRate.append(trained_data[i].tearRate)
        self.col_diabetic.append(trained_data[i].diabetic)
        self.col_needLense.append(trained_data[i].needLense)
```

```python
def entropy(self, column_of_features):
    values_zero = 0
    values_one = 0
    entropy = 0
    for i in range(0,len(column_of_features)):
        if column_of_features[i]==0:
            values_zero+=1
        elif column_of_features[i]==1:
            values_one+=1
    length=len(column_of_features)
    #print("l  = ", length)
```

```python
    if (values_zero/length) == 0 and (values_one/length) != 0:
        entropy = -(0 + ((values_one / length) * math.log2(values_one / length)))
    elif values_zero/length != 0 and values_one/length == 0:
        entropy = -(((values_zero / length) * math.log2(values_zero / length))  + 0)
    elif values_zero / length == 0 and values_one / length == 0:
        entropy = 0
    else:
        entropy = -(((values_zero / length) * math.log2(values_zero / length)) +
                    ((values_one / length) * math.log2(values_one / length)))

    return entropy
```

```python
def gain(self, column_of_features, needLense_desicion_col):
    #print("start gain")
    length_features = len(column_of_features)
    zero_feature_count = 0
    one_feature_count = 0
    zero_feature_list = []
    one_feature_list = []
    for i in range(0, length_features):
        if column_of_features[i] == 0:
            zero_feature_count += 1
            zero_feature_list.append(needLense_desicion_col[i])
        elif column_of_features[i] == 1:
            one_feature_count += 1
            one_feature_list.append(needLense_desicion_col[i])
```

```python
        #print("-->", one_feature_list)
Total_entropy = self.entropy(needLense_desicion_col)
zero_feature_entropy= self.entropy(zero_feature_list)
one_feature_entropy = self.entropy(one_feature_list)
gain_feature = Total_entropy - ( ((zero_feature_count / length_features) * zero_feature_entropy)
                                + ((one_feature_count / length_features) * one_feature_entropy) )
return gain_feature
```

```python
    def classify(self, input):
        # takes an array for the features ex. [0, 0, 1, 1, 1]
        # should return 0 or 1 based on the classification
        for i in range(0, len(self.features)):
            self.features[i].visited = -1

        self.set_all_data()
        self.id3_main_algorithim(2)
        global N
        for i in self.dictionary_nodes:
            N = self.dictionary_nodes[i]
            break
        while True:
            if N.name_node == 'diabetic':
                if input[4] == 1:
                    if N.right_node == 0 or N.right_node == 1:
                        return N.right_node
                    else:
```

```python
                    else:
                        N = self.dictionary_nodes.get(N.right_node)
                elif input[3] == 0:
                    if N.left_node == 0 or N.left_node == 1:
                        return N.left_node
                    else:
                        N = self.dictionary_nodes.get(N.left_node)
            if N.name_node == 'tearRate':
                if input[3] == 1:
                    if N.right_node == 0 or N.right_node == 1:
                        return N.right_node
                    else:
                        N = self.dictionary_nodes.get(N.right_node)
                elif input[3] == 0:
                    if N.left_node == 0 or N.left_node == 1:
                        return N.left_node
                    else:
                        N = self.dictionary_nodes.get(N.left_node)
```

```
elif N.name_node == 'astigmatic':
    if input[2] == 1:
        if N.right_node == 0 or N.right_node == 1:
            return N.right_node
        else:
            N = self.dictionary_nodes.get(N.right_node)
    elif input[2] == 0:
        if N.left_node == 0 or N.left_node == 1:
            return N.left_node
        else:
            N = dictionary_nodes.get(N.left_node)
```

```
elif N.name_node == 'age':
    if input[0] == 1:
        if N.right_node == 0 or N.right_node == 1:
            return N.right_node
        else:
            N = self.dictionary_nodes.get(N.right_node)
    elif input[0] == 0:
        if N.left_node == 0 or N.left_node == 1:
            return N.left_node
        else:
            N = self.dictionary_nodes.get(N.left_node)
```

```
elif N.name_node == 'prescription':
    if input[1] == 1:
        if N.right_node == 0 or N.right_node == 1:
            return N.right_node
        else:
            N = self.dictionary_nodes.get(N.right_node)
    elif input[1] == 0:
        if N.left_node == 0 or N.left_node == 1:
            return N.left_node
        else:
            N = dictionary_nodes.get(N.left_node)
```

2.3.3 Sample run (the output)

- not based complete
- But as I expected the output would be 1, 0, 0, 1

3. Discussion:

1. Breadth First Search:-

- **Efficient**
  - The complexity is O(M * N).
    - M: no. of columns.
    - N: no. of rows.
- **Improving**
  - For improving may be the complexity O(M+N).

2. Neural Network:-

- **Efficient**
  - The Code cover all test cases about AND logic gate.
- **Improving**
  - Update basic factor after each iteration to decreasing the number of each iteration.
  - Find the correct solution after N iterations.
    - N: the length of trained inputs.

3. ID3:-

- **Efficient**
  - The accuracy of prediction is more than (86%).
- **Improving**
  - Find the answer of all cases.

4. References :-

[1] Ma, G.: Development and Application Study of Hospital Management Information System Based on Data Mining Technology, Xi'an Shiyou University (2008).

[2] L.Surya Prasanthi et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (6) , 2015

[3] Ahmed Salah. "algorithms Graphs.", AinShams university, 2019

[4] T.A. "AI Course. Lab 3 BFS", AinShams university, 2020

[5] T.A. "AI Course. Lab 6 ID3", AinShams university, 2020

[6] T.A. "AI Course. Lab 11 ANN", AinShams university, 2020

[7] Marco alfons. "AI Course. Lec 10 ANN", AinShams university, 2020