AStarSearchAlgorithm

January 4, 2022

[ ]:
```python
def astarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}
    parents = {}

    g[start_node] = 0
    parents[start_node] = start_node

    while len(open_set) > 0:
        n = None

        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v

        if n ==  stop_node or graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbours(n):
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                else:
                    if g[m] > g[n] + weight:
                        g[m] = g[n] + weight
                        parents[m] = n

                        if m in closed_set:
                            closed_set.remove(m)
                            open_set.add(m)

        if n == None:
            print('Path Doesn\'t Exist!')
            return None
```

```python
        if n == stop_node:
            path = []

            while parents[n] != n:
                path.append(n)
                n = parents[n]

            path.append(start_node)
            path.reverse()

            print('Path Found : ', format(path))
            return path

        open_set.remove(n)
        closed_set.add(n)

    print('Path Doesn\'t Exist!')
    return None
```

[ ]:
```python
def get_neighbours(v):
    if v in graph_nodes:
        return graph_nodes[v]
    else:
        return None
```

[ ]:
```python
def heuristic(n):
    H_dist = {
        'A' : 10,
        'B' : 8,
        'C' : 5,
        'D' : 7,
        'E' : 3,
        'F' : 6,
        'G' : 5,
        'H' : 4,
        'I' : 1,
        'J' : 0
    }
    return H_dist[n]
```

[ ]:
```python
graph_nodes = {
    'A' : [('B',6),('F',3)],
    'B' : [('C',3),('D',2)],
    'C' : [('D',1),('E',5)],
    'D' : [('C',1),('E',8)],
    'E' : [('I',5),('J',5)],
    'F' : [('G',1),('H',7)],
```

```
    'G' : [('I',3)],
    'H' : [('I',2)],
    'I' : [('E',5),('J',3)]
}
```

[ ]:
```
astarAlgo('A','J')
```

```
Path Found :  ['A', 'F', 'G', 'I', 'J']
```

[ ]:
```
['A', 'F', 'G', 'I', 'J']
```

AOStarSearchAlgorithm

January 4, 2022

[ ]:
```
def recAOStar(n):
    global finalPath
    print("Expanding Node :", n)
    and_nodes = []
    or_nodes = []

    if (n in allNodes):
        if 'AND' in allNodes[n]:
            and_nodes = allNodes[n]['AND']
        if 'OR' in allNodes[n]:
            or_nodes = allNodes[n]['OR']
    if len(and_nodes) == 0 and len(or_nodes) == 0:
        return
    solvable = False
    marked = {}

    while not solvable:

        if len(marked) == len(and_nodes) + len(or_nodes):
            min_cost_least, min_cost_group_least = least_cost_group(and_nodes,␣
↪or_nodes, {})
            solvable = True
            change_heuristic(n, min_cost_least)
            optimal_child_group[n] = min_cost_group_least
            continue

        min_cost, min_cost_group = least_cost_group(and_nodes, or_nodes, marked)
        is_expanded = False


        if len(min_cost_group) > 1:
            if (min_cost_group[0] in allNodes):
                is_expanded = True
                recAOStar(min_cost_group[0])
            if (min_cost_group[1] in allNodes):
                is_expanded = True
                recAOStar(min_cost_group[1])
```

```
        else:
            if (min_cost_group in allNodes):
                is_expanded = True
                recAOStar(min_cost_group)

        if is_expanded:
            min_cost_verify, min_cost_group_verify =␣
↪least_cost_group(and_nodes, or_nodes, {})
            if min_cost_group == min_cost_group_verify:
                solvable = True
                change_heuristic(n, min_cost_verify)
                optimal_child_group[n] = min_cost_group

        else:
              solvable = True
              change_heuristic(n, min_cost)
              optimal_child_group[n] = min_cost_group

        marked[min_cost_group] = 1
    return heuristic(n)
```

[ ]:
```
def least_cost_group(and_nodes, or_nodes, marked):
    node_wise_cost = {}

    for node_pair in and_nodes:
        if not node_pair[0] + node_pair[1] in marked:
            cost = 0
            cost = cost + heuristic(node_pair[0]) + heuristic(node_pair[1]) + 2
            node_wise_cost[node_pair[0] + node_pair[1]] = cost
    for node in or_nodes:
        if not node in marked:
            cost = 0
            cost = cost + heuristic(node) + 1
            node_wise_cost[node] = cost

    min_cost = 999999
    min_cost_group = None

    for costKey in node_wise_cost:
        if node_wise_cost[costKey] < min_cost:
            min_cost = node_wise_cost[costKey]
            min_cost_group = costKey
    return [min_cost, min_cost_group]
```

[ ]:
```
def heuristic(n):
    return H_dist[n]
```

[ ]:
```python
def change_heuristic(n, cost):
    H_dist[n] = cost
    return
```

[ ]:
```python
def print_path(node):
    print(optimal_child_group[node], end="")
    node = optimal_child_group[node]

    if len(node) > 1:
          if node[0] in optimal_child_group:
              print("->", end="")
              print_path(node[0])
          if node[1] in optimal_child_group:
              print("->", end="")
              print_path(node[1])
    else:
          if node in optimal_child_group:
              print("->", end="")
              print_path(node)
```

[ ]:
```python
H_dist = {
    'A': -1,
    'B': 4,
    'C': 2,
    'D': 3,
    'E': 6,
    'F': 8,
    'G': 2,
    'H': 0,
    'I': 0,
    'J': 0
}
```

[ ]:
```python
allNodes = {
      'A': {'AND': [('C', 'D')], 'OR': ['B']},
      'B': {'OR': ['E', 'F']},
      'C': {'AND': [('H', 'I')], 'OR': ['G']},
      'D': {'OR': ['J']}
}
```

[ ]:
```python
optimal_child_group = {}
optimal_cost = recAOStar('A')

print('Nodes Which Gives Optimal Cost Are')
print_path('A')
print('\nOptimal Cost Is :', optimal_cost)
```

```
Expanding Node : A
Expanding Node : B
Expanding Node : C
Expanding Node : D
Nodes Which Gives Optimal Cost Are
CD->HI->J
Optimal Cost Is : 5
```

CandidateEliminationAlgorithm

January 4, 2022

[ ]:
```python
import pandas as pd
df = pd.read_csv('Datasets/EnjoySports.csv')
# df = df.drop(['slno'], axis = 1)
concepts = df.values[:,:-1]
target = df.values[:,-1]
df.head()
```

[ ]:

	Sky	AirTemp	Humidity	Wind	Water	Forecast	Enjoysport
0	sunny	warm	normal	strong	warm	same	yes
1	sunny	warm	high	strong	warm	same	yes
2	rainy	cold	high	strong	warm	change	no
3	sunny	warm	high	strong	cool	change	yes

[ ]:
```python
def learn(concepts,target):
    specific_h = concepts[0].copy()
    general_h = [["?" for i in range(len(specific_h))] for i in␣
 ↪range(len(specific_h))]

    for i,h in enumerate(concepts):
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'

        if target[i] == "no":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

    indices = [i for i,val in enumerate(general_h)
    if val == ['?','?','?','?','?','?']]
    for i in indices:
        general_h.remove(['?','?','?','?','?','?'])
    return specific_h,general_h
```

[ ]:
```python
s_final, g_final = learn(concepts,target)
print(f" Final S : {s_final}")
print(f" Final G : {g_final}")
```

```
 Final S : ['sunny' 'warm' '?' 'strong' '?' '?']
 Final G : [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?',
'?']]
```

DecisionTreeAlgorithm

January 4, 2022

[ ]:
```python
def infoGain(P, N):                                         # Calculate Information␣
 ↪Gain Or Class Entropy
    import math
    return -P / (P + N) * math.log2(P / (P + N)) - N / (P + N) * math.log2(N /␣
 ↪(P + N))
```

[ ]:
```python
def insertNode(tree, addTo, Node):
    for k, v in tree.items():                               # Traversal Of Tree
        if isinstance(v, dict):
            tree[k] = insertNode(v, addTo, Node)
    if addTo in tree:                                       # If d Is Found then Add␣
 ↪Node
        if isinstance(tree[addTo], dict):
            tree[addTo][Node] = 'None'
        else:
            tree[addTo] = {Node: 'None'}
    return tree
```

[ ]:
```python
def insertConcept(tree, addTo, Node):
    for k, v in tree.items():                               # Traversal Of Tree
        if isinstance(v, dict):
            tree[k] = insertConcept(v, addTo, Node)
    if addTo in tree:                                       # If d Is Found Then Add␣
 ↪Node
        tree[addTo] = Node
    return tree
```

[ ]:
```python
def getNextNode(data, AttributeList, concept, conceptVals, tree, addTo):
    Total = data.shape[0]
    if Total == 0:                                                              ␣
 ↪          # If Attibutes Are Empty, THen Return Current Value Of Tree
        return tree

    countC = {}
    for cVal in conceptVals:                                                    ␣
 ↪          # If Example Is Positive, Then Return Positive And If Negative,␣
 ↪Then Return Negative
```

```python
        dataCC = data[data[concept] == cVal]            # Get Data For Specific Concept
        countC[cVal] = dataCC.shape[0]            # Get The Count Of Data For Specific Concept

    if countC[conceptVals[0]] == 0:            # If All Examples Are Positive (Not Negative), Return Single Node Positive
        tree = insertConcept(tree, addTo, conceptVals[1])
        return tree
    if countC[conceptVals[1]] == 0:            # If All Examples Are Negative (Not Positive), Return Single Node Negative
        tree = insertConcept(tree, addTo, conceptVals[0])
        return tree

    ClassEntropy = infoGain(countC[conceptVals[0]], countC[conceptVals[1]])            # Calculate Class Entropy For Data

    Attr = {}            # Attribute Dictionary Holding List Of Possible Values
    for a in AttributeList:
        Attr[a] = list(set(data[a]))

    AttrCount = {}            # Get The Attribute Values Being Positive And Negative
    EntropyAttr = {}            # Attribute Entropy
    for att in Attr:
        for vals in Attr[att]:
            for c in conceptVals:
                iData = data[data[att] == vals]            # Get Data For Specific Attribute
                dataAtt = iData[iData[concept] == c]            # Get Data For Specific Attribute And Concept
                AttrCount[c] = dataAtt.shape[0]            # Get The Count Of Data For Specific Attribute And Concept
            TotalInfo = AttrCount[conceptVals[0]] + AttrCount[conceptVals[1]]            # Total Attribute
            if AttrCount[conceptVals[0]] == 0 or AttrCount[conceptVals[1]] == 0:
                InfoGain = 0
            else:
                InfoGain = infoGain(AttrCount[conceptVals[0]], AttrCount[conceptVals[1]])  # Calculate InfoGain For Each Attr
```

```
            if att not in EntropyAttr:                                            ␣
↪             # Calculate Entropy For Each Attr
                EntropyAttr[att] = (TotalInfo / Total) * InfoGain
            else:
                EntropyAttr[att] = EntropyAttr[att] + (TotalInfo / Total) *␣
↪InfoGain

    Gain = {}
    for g in EntropyAttr:
        Gain[g] = ClassEntropy - EntropyAttr[g]                                ␣
↪             # Calculate Gain

    Node = max(Gain, key=Gain.get)                                             ␣
↪             # Get Root Node

    tree = insertNode(tree, addTo, Node)                                       ␣
↪             # Add Node To Tree
    for nD in Attr[Node]:
        tree = insertNode(tree, Node, nD)                                      ␣
↪             # Insert Attribute Value To Tree
        newData = data[data[Node] == nD].drop(Node, axis=1)                    ␣
↪             # Get New Data With Attribute Value nD And Removing Rows With␣
↪Column Value Node
        AttributeList = list(newData)[:-1]                                     ␣
↪             # New Attribute List
        tree = getNextNode(newData, AttributeList, concept, conceptVals, tree,␣
↪nD)          # Call The Function Recursively
    return tree
```

[ ]:

```
import pandas as pd
def main():
    data = pd.read_csv('Datasets/PlayTennis.csv')                     # Reading␣
↪CSV
    if 'Unnamed: 0' in data.columns:
        data = data.drop('Unnamed: 0', axis=1)
#       data = data.drop('slno', axis=1)
    AttributeList = list(data)[:-1]                                   # Get␣
↪Attribute List
    concept = str(list(data)[-1])                                     # Get␣
↪Concept List
    conceptVals = list(set(data[concept]))                            # Get␣
↪Concept Values
    tree = getNextNode(data, AttributeList, concept, conceptVals,
                       {'root': 'None'}, 'root')
    return tree                                                       # Call␣
↪Recursive Function With Initial Value Of Tree And Node As Root
```

[ ]:
```
tree = main()['root']
```

[ ]:
```
df = pd.read_csv('Datasets/PlayTennis.csv')

def test(tree, d):
    for k in tree:
        for v in tree[k]:
            if (d[k] == v and isinstance(tree[k][v], dict)):
                test(tree[k][v], d)
            elif (d[k] == v):
                print("Classification: " + tree[k][v])
```

[ ]:
```
if 'Unnamed: 0' in df.columns:
    df = df.drop('Unnamed: 0', axis=1)
df.head()
```

[ ]:

	Outlook	Temperature	Humidity	Wind	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes

[ ]:
```
print(tree)
test(tree, df.loc[0, :])
```

```
{'Outlook': {'Overcast': 'Yes', 'Sunny': {'Humidity': {'Normal': 'Yes', 'High':
'No'}}, 'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}}}}
Classification: No
```

ANNBackPropagationAlgorithm

January 4, 2022

[ ]:
```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float) # Two Inputs [sleep, study]
y = np.array(([92], [86], [89]), dtype=float)       # One Output [Expected % In␣
 ↪Exams]
X = X / np.amax(X, axis=0)                          # Maximum Of X Array␣
 ↪Longitudinally
y = y / 100
```

[ ]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

[ ]:
```
def derivatives_sigmoid(x):
    return x * (1 - x)
```

[ ]:
```
epoch = 1000                # Number Of Iterations
learning_rate = 0.6         # Learning Rate ETA
inputlayer_neurons = 2      # Number Of Neurons In Input Layer
hiddenlayer_neurons = 3     # Number Of Neurons In Hidden Layer
output_neurons = 1          # Number Of Neurons In Output Layer
```

[ ]:
```
wh = np.random.uniform(size = (inputlayer_neurons,
                             hiddenlayer_neurons))       # wh = Hidden Layer␣
 ↪Weights
bh = np.random.uniform(size = (1, hiddenlayer_neurons))  # bh = Hidden Layer␣
 ↪Bias
wo = np.random.uniform(size = (hiddenlayer_neurons,
                             output_neurons))            # wo = Output Layer␣
 ↪Weights
bo = np.random.uniform(size = (1, output_neurons))       # bo = Output Layer␣
 ↪Bias
```

[ ]:
```
for i in range(epoch):

    # Forward Propogation
    net_h = np.dot(X, wh) + bh                                 # Net Input For␣
 ↪Hidden Layer
```

```
    sigma_h = sigmoid(net_h)                                # Output Of␣
↪sigmoid Function Of Hidden Layer
    net_o = np.dot(sigma_h, wo) + bo                        # Net Input For␣
↪Output Layer
    output = sigmoid(net_o)                                 # The Output Of␣
↪Output Layer i.e sigmoid Of net_o

    # Back Propagation
    deltaK = (y - output) * derivatives_sigmoid(output)     # Calculate deltak
    deltaH = deltaK.dot(wo.T) * derivatives_sigmoid(sigma_h)  # deltaH
    wo = wo + sigma_h.T.dot(deltaK) * learning_rate         # Update Output␣
↪Layer Weights
    wh = wh + X.T.dot(deltaH) * learning_rate               # Update Hidden␣
↪Layer Weights
```

[ ]:
```
print ("Input: \n" + str(X))
print ("Actual Output: \n" + str(y))
print ("Predicted Output: \n", output)
```

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89347243]
 [0.88144644]
 [0.89533573]]
```

NaïveBayesianClassifier

January 4, 2022

[ ]:
```python
def probAttr(data, attr, val):
    Total = data.shape[0]                   # Get Column Length
    cnt = len(data[data[attr] == val])     # Count Of Attribute [attr] Equal To␣
 ↪Val
    return cnt, cnt / Total
```

[ ]:
```python
def train(data, Attr, conceptVals, concept):
    conceptProbs = {}                                                           ␣
 ↪                              # P(A)
    countConcept = {}
    for cVal in conceptVals:                                                    ␣
 ↪                              # Get Probablity And Count Of Yes And No
        countConcept[cVal], conceptProbs[cVal] = probAttr(data, concept, cVal)

    AttrConcept = {}                                                            ␣
 ↪                              # P(X/A)
    probability_list = {}                                                       ␣
 ↪                              # P(X)
    for att in Attr:                                                            ␣
 ↪                              # Create A Tree For Attribute
        probability_list[att] = {}
        AttrConcept[att] = {}
        for val in Attr[att]:                                                   ␣
 ↪                              # Create A Tree For Attribute Value
            AttrConcept[att][val] = {}
            a,probability_list[att][val] = probAttr(data, att, val)             ␣
 ↪                                # Get Probablity For att Equal To val
            for cVal in conceptVals:                                            ␣
 ↪                              # Create A Tree To Hold Yes And No Values
                dataTemp = data[data[att]==val]                                 ␣
 ↪                              # Calculate att Equal To val and concept␣
 ↪Equal To cVal
                AttrConcept[att][val][cVal] = len(dataTemp[dataTemp[concept] ==␣
 ↪cVal]) / countConcept[cVal]

    print("P(A) : ", conceptProbs,"\n")
    print("P(X/A) : ", AttrConcept,"\n")
```

```
    print("P(X) : ", probability_list,"\n")
    return conceptProbs, AttrConcept, probability_list
```

[ ]:
```
def test(examples, Attr, concept_list, conceptProbs, AttrConcept, probability_list):
    misclassification_count = 0
    Total = len(examples)                       # Get Number Of Testing Set
    for ex in examples:
        px = {}                                 # Dictionary To Hold Final Value
        for a in Attr:                          # Iterrate Thorugh The Tree With Attributes
            for x in ex:                        # Iterrate Thorugh The Tree For Given Example
                for c in concept_list:          # Iterrate Thorugh The Tree Using Concepts
                    if x in AttrConcept[a]:     # Check If The Value Of x Refering In Same Sub-Tree Of P(X/A)
                        if c not in px:         # If c Not In px Multiply P(A) With 1st Itteration (For 1st Value Of x)
                            px[c] = conceptProbs[c] * AttrConcept[a][x][c] / probability_list[a][x]
                        else:                   # Multiply px In Next Itterations (For Next Value Of x)
                            px[c] = px[c] * AttrConcept[a][x][c] / probability_list[a][x]
        print(px)
        classification = max(px, key = px.get)  # Key Of Maximum Of px Is Required Classification
        print("Classification :", classification, "Expected :", ex[-1])
        if(classification != ex[-1]):
            misclassification_count += 1
    misclassification_rate = misclassification_count * 100 / Total
    accuracy = 100 - misclassification_rate
    print("Misclassification Count = {}".format(misclassification_count))
    print("Misclassification Rate = {}%".format(misclassification_rate))
    print("Accuracy = {}%".format(accuracy))
```

[ ]:
```
import pandas as pd
data = pd.read_csv('Datasets/PlayTennis.csv')
data.drop(['Unnamed: 0'], axis = 1, inplace = True)
print(data)
concept = str(list(data)[-1])
```

```
print(concept)
concept_list = set(data[concept])
print(concept_list)
Attr = {}
for a in list(data)[:-1]:
    Attr[a] = set(data[a])
    print(Attr[a])
conceptProbs, AttrConcept, probability_list = train(data, Attr, concept_list,␣
 ↪concept)

examples = pd.read_csv('Datasets/PlayTennis.csv')
test(examples.values, Attr, concept_list, conceptProbs, AttrConcept,␣
 ↪probability_list)
```

```
     Outlook Temperature Humidity    Wind PlayTennis
0      Sunny         Hot     High    Weak         No
1      Sunny         Hot     High  Strong         No
2   Overcast         Hot     High    Weak        Yes
3       Rain        Mild     High    Weak        Yes
4       Rain        Cool   Normal    Weak        Yes
5       Rain        Cool   Normal  Strong         No
6   Overcast        Cool   Normal  Strong        Yes
7      Sunny        Mild     High    Weak         No
8      Sunny        Cool   Normal    Weak        Yes
9       Rain        Mild   Normal    Weak        Yes
10     Sunny        Mild   Normal  Strong        Yes
11  Overcast        Mild     High  Strong        Yes
12  Overcast         Hot   Normal    Weak        Yes
13      Rain        Mild     High  Strong         No
PlayTennis
{'No', 'Yes'}
{'Overcast', 'Sunny', 'Rain'}
{'Hot', 'Mild', 'Cool'}
{'Normal', 'High'}
{'Strong', 'Weak'}
P(A) :  {'No': 0.35714285714285715, 'Yes': 0.6428571428571429}

P(X/A) :  {'Outlook': {'Overcast': {'No': 0.0, 'Yes': 0.4444444444444444},
'Sunny': {'No': 0.6, 'Yes': 0.2222222222222222}, 'Rain': {'No': 0.4, 'Yes':
0.3333333333333333}}, 'Temperature': {'Hot': {'No': 0.4, 'Yes':
0.2222222222222222}, 'Mild': {'No': 0.4, 'Yes': 0.4444444444444444}, 'Cool':
{'No': 0.2, 'Yes': 0.3333333333333333}}, 'Humidity': {'Normal': {'No': 0.2,
'Yes': 0.6666666666666666}, 'High': {'No': 0.8, 'Yes': 0.3333333333333333}},
'Wind': {'Strong': {'No': 0.6, 'Yes': 0.3333333333333333}, 'Weak': {'No': 0.4,
'Yes': 0.6666666666666666}}}

P(X) :  {'Outlook': {'Overcast': 0.2857142857142857, 'Sunny':
```

```
0.35714285714285715, 'Rain': 0.35714285714285715}, 'Temperature': {'Hot':
0.2857142857142857, 'Mild': 0.42857142857142855, 'Cool': 0.2857142857142857},
'Humidity': {'Normal': 0.5, 'High': 0.5}, 'Wind': {'Strong':
0.42857142857142855, 'Weak': 0.5714285714285714}}

{'No': 0.9408000000000002, 'Yes': 0.2419753086419753}
Classification : No Expected : No
{'No': 1.8816000000000002, 'Yes': 0.16131687242798354}
Classification : No Expected : No
{'No': 0.0, 'Yes': 0.6049382716049383}
Classification : Yes Expected : Yes
{'No': 0.4181333333333335, 'Yes': 0.4839506172839506}
Classification : Yes Expected : Yes
{'No': 0.07840000000000004, 'Yes': 1.0888888888888888}
Classification : Yes Expected : Yes
{'No': 0.15680000000000005, 'Yes': 0.7259259259259259}
Classification : Yes Expected : No
{'No': 0.0, 'Yes': 1.2098765432098766}
Classification : Yes Expected : Yes
{'No': 0.6272000000000001, 'Yes': 0.3226337448559671}
Classification : No Expected : No
{'No': 0.11760000000000002, 'Yes': 0.7259259259259256}
Classification : Yes Expected : Yes
{'No': 0.10453333333333338, 'Yes': 0.9679012345679012}
Classification : Yes Expected : Yes
{'No': 0.31360000000000005, 'Yes': 0.43017832647462273}
Classification : Yes Expected : Yes
{'No': 0.0, 'Yes': 0.5377229080932785}
Classification : Yes Expected : Yes
{'No': 0.0, 'Yes': 1.2098765432098766}
Classification : Yes Expected : Yes
{'No': 0.8362666666666669, 'Yes': 0.3226337448559671}
Classification : No Expected : No
Misclassification Count = 1
Misclassification Rate = 7.142857142857143%
Accuracy = 92.85714285714286%
```

EMandKMeansAlgorithm

January 4, 2022

[ ]:
```python
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
```

[ ]:
```python
iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
y = pd.DataFrame(iris.target)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y.columns = ['Targets']


model = KMeans(n_clusters = 3)

model.fit(X)
```

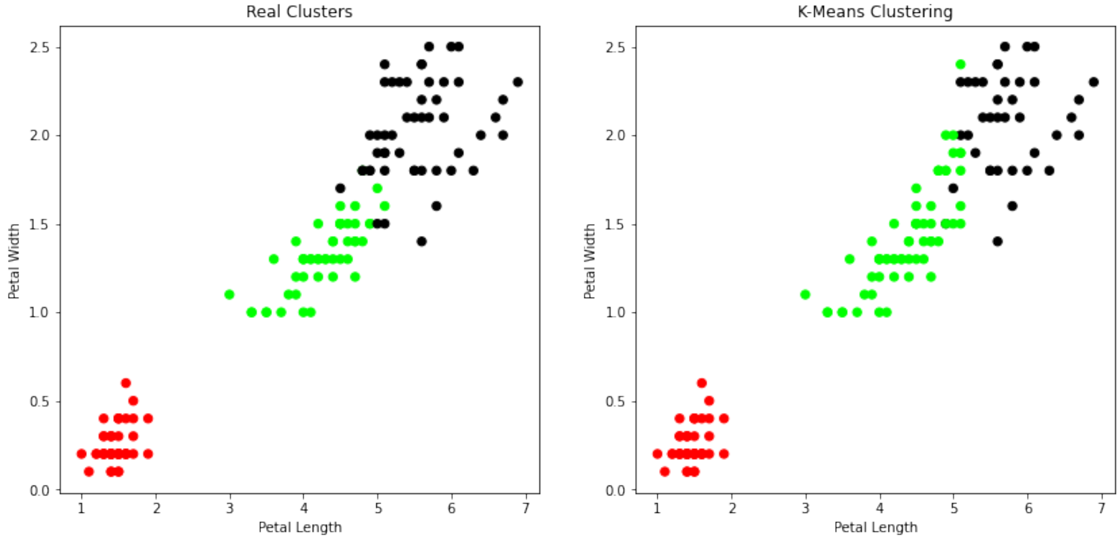
[ ]:
```
KMeans(n_clusters=3)
```

[ ]:
```python
plt.figure(figsize = (14, 14))
colormap = np.array(['red', 'lime', 'black'])

plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c = colormap[y.Targets], s = 40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c = colormap[model.labels_], s = 40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

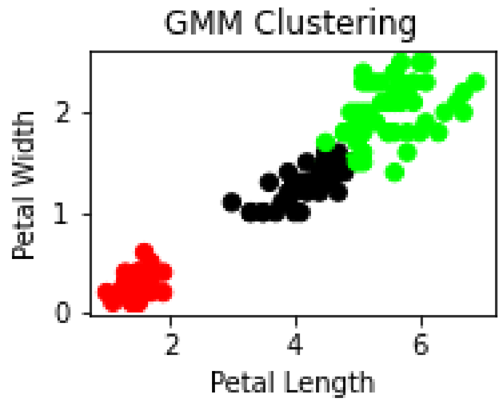
[ ]:
```
Text(0, 0.5, 'Petal Width')
```




```
[ ]: from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
```

```
[ ]: from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components = 3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)
plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c = colormap[gmm_y], s = 40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('Observation: The GMM Using EM Algorithm Based Clustering Matched The␣
↪True Labels More Closely Than The K-Means.')
```

```
Observation: The GMM Using EM Algorithm Based Clustering Matched The True Labels
More Closely Than The K-Means.
```


GMM Clustering
Petal Width
Petal Length
0
1
2
2
4
6

KNearestNeighbourAlgorithm

January 4, 2022

[ ]:
```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import numpy as np
```

[ ]:
```
dataset = load_iris()
X_train,X_test,y_train,y_test =␣
 ↪train_test_split(dataset["data"],dataset["target"],random_state = 0)
```

[ ]:
```
kn = KNeighborsClassifier(n_neighbors = 3)
kn.fit(X_train,y_train)
```

[ ]:
```
KNeighborsClassifier(n_neighbors=3)
```

[ ]:
```
prediction = kn.predict(X_test)
confusion_matrix(y_test, prediction)
```

[ ]:
```
array([[13,  0,  0],
       [ 0, 15,  1],
       [ 0,  0,  9]], dtype=int64)
```

LocallyWeightedRegressionalAlgorithm

January 4, 2022

[ ]:
```
from math import ceil
import numpy as np
from scipy import linalg
```

[ ]:
```
def lowess(x, y, f, iterations):
    n = len(x)
    r = int(ceil(f * n))
    h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]
    w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)
    w = (1 - w ** 3) ** 3
    yest = np.zeros(n)
    delta = np.ones(n)
    for iteration in range(iterations):
        for i in range(n):
            weights = delta * w[:, i]
            b = np.array([np.sum(weights * y), np.sum(weights * y * x)])
            A = np.array([[np.sum(weights), np.sum(weights * x)],[np.
 ↪sum(weights * x), np.sum(weights * x * x)]])
            beta = linalg.solve(A, b)
            yest[i] = beta[0] + beta[1] * x[i]

        residuals = y - yest
        s = np.median(np.abs(residuals))
        delta = np.clip(residuals / (6.0 * s), -1, 1)
        delta = (1 - delta ** 2) ** 2

    return yest
```

[ ]:
```
def main():
    import math
    n = 100
    x = np.linspace(0, 2 * math.pi, n)
    y = np.sin(x) + 0.3 * np.random.randn(n)
    f = 0.25
    iterations = 3
    yest = lowess(x, y, f, iterations)
```

```python
import matplotlib.pyplot as plt
plt.plot(x,y,"r.")
plt.plot(x,yest,"b-")
```

[ ]:
```python
main()
```

```
Matplotlib is building the font cache; this may take a moment.
```