ai-lab-code

March 17, 2023

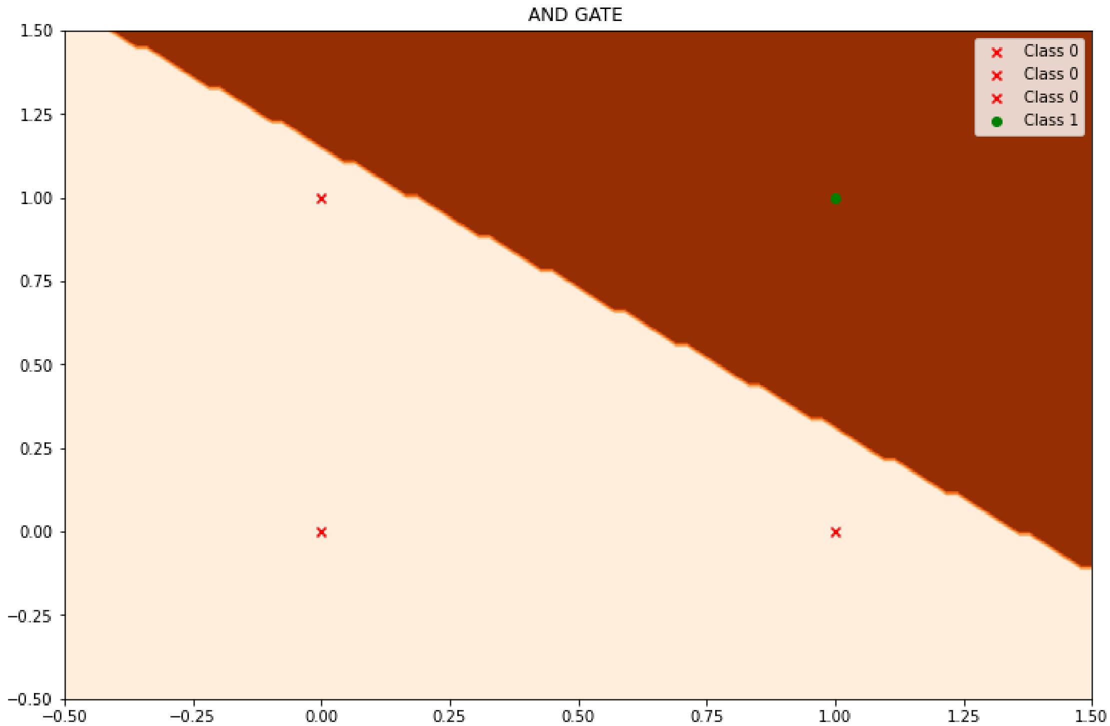
1 AND Gate Single layer Perceptron

[1]:
```python
import numpy as np
def activation(out, threshold):
    act_out = 1/(1 + np.exp(-out))
    if act_out>=threshold:
        return 1
    else:
        return 0

def perceptron(and_input):
    a = [0, 0, 1, 1]
    b = [0, 1, 0, 1]
    y = [0, 0, 0, 1]
    bias = -1
    w = [0.6, 0.8, 1.5]
    threshold = 0.5
    learning_rate = 0.5
    i = 0
    print("Perceptron Training")
    print("####################")
    print("________________")
    while i<4:
        summation = (a[i]*w[0] + b[i]*w[1]) + bias*w[2]
        target_output = activation(summation, threshold)
        print("Input : " + str(a[i]) + " , "+str(b[i]))
        print("Weights : " + str(w[0]) + " , "+str(w[1]))
        print("Summation : " + str(summation))
        print("Actual output : " + str(y[i]) + " Predicted Output␣
 ↪"+str(target_output))

        if(target_output != y[i]):
            print(".........\nUpdating Weights")
            w[0] = w[0] + learning_rate*(y[i] - target_output)*a[i]
            w[1] = w[1] + learning_rate*(y[i] - target_output)*b[i]
            print("Updated Weights: ", str(w[0]) + ','+str(w[1]))
            i = -1
```

```
            print("\nWeights Updated Training Again : ")
            print("####################################")
        i = i+1
        #print("----------------")
    summation = and_input[0]*w[0] + and_input[1]*w[1] + bias*w[2]
    import matplotlib.pyplot as plt
    plt.figure(figsize=(12, 8))
    x_min, x_max = -0.5, 1.5
    y_min, y_max = -0.5, 1.5
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min,␣
↪y_max, 100))
    Z = np.array([activation((np.dot([w[0],w[1]], [a, b]) + bias*w[2]),␣
↪threshold) for a, b in np.c_[xx.ravel(), yy.ravel()]])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap='Oranges')
    for i in range(len(y)):
        if y[i] == 0:
            plt.scatter(a[i], b[i], color='red', marker='x', label='Class 0')
        else:
            plt.scatter(a[i], b[i], color='green', marker='o', label='Class 1')
    #plt.scatter(a,b, c=y, cmap = "Blues_r", label = "a & b values")
    plt.title("AND GATE ")
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    #plt.xticks(())
    #plt.yticks(())
    #plt.grid()
    plt.legend()
    plt.show()
    return activation(summation, threshold)

and_input = [1,1]
output = perceptron(and_input)
print("AND Gate Output for "+ str(and_input) + " : " + str(output))
```

```
Perceptron Training
####################

----------------
Input : 0 , 0
Weights : 0.6 , 0.8
Summation : -1.5
Actual output : 0 Predicted Output 0
Input : 0 , 1
Weights : 0.6 , 0.8
Summation : -0.7
Actual output : 0 Predicted Output 0
Input : 1 , 0
```

```
Weights : 0.6 , 0.8
Summation : -0.9
Actual output : 0 Predicted Output 0
Input : 1 , 1
Weights : 0.6 , 0.8
Summation : -0.10000000000000009
Actual output : 1 Predicted Output 0
…
Updating Weights
Updated Weights:  1.1,1.3

Weights Updated Training Again :
##################################
Input : 0 , 0
Weights : 1.1 , 1.3
Summation : -1.5
Actual output : 0 Predicted Output 0
Input : 0 , 1
Weights : 1.1 , 1.3
Summation : -0.19999999999999996
Actual output : 0 Predicted Output 0
Input : 1 , 0
Weights : 1.1 , 1.3
Summation : -0.3999999999999999
Actual output : 0 Predicted Output 0
Input : 1 , 1
Weights : 1.1 , 1.3
Summation : 0.9000000000000004
Actual output : 1 Predicted Output 1
```




```
AND Gate Output for [1, 1] : 1
```

2 OR Gate Single layer Perceptron

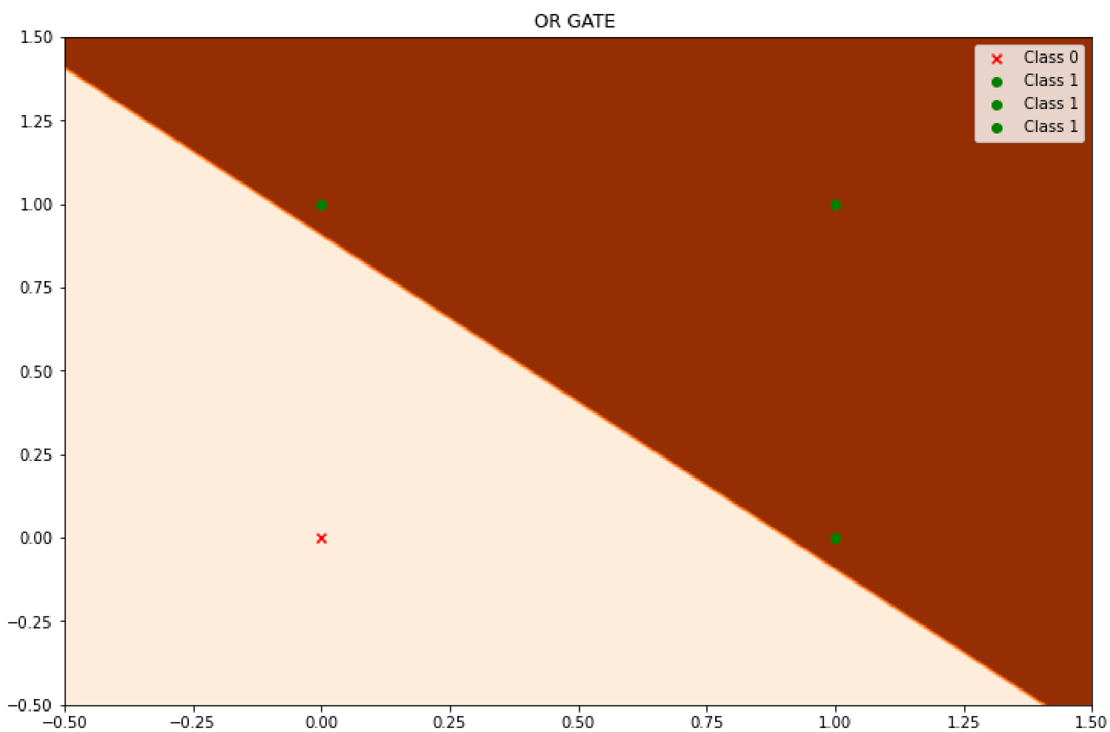
[2]:
```python
import numpy as np
def activation(out, threshold):
    act_out = 1/(1 + np.exp(-out))
    if act_out>=threshold:
        return 1
    else:
        return 0
def perceptron(and_input):
    a = [0, 0, 1, 1]
    b = [0, 1, 0, 1]
    y = [0, 1, 1, 1]
    bias = -1
    w = [0.6,0.6,1]
    threshold = 0.5
    learning_rate = 0.5
    i = 0
    print("Perceptron Training")
    print("####################")
    print("________________")
```

```
    while i<4:
        summation = (a[i]*w[0] + b[i]*w[1]) + bias*w[2]
        target_output = activation(summation, threshold)
        print("Input : " + str(a[i]) + " , "+str(b[i]))
        print("Weights : " + str(w[0]) + " , "+str(w[1]))
        print("Summation : " + str(summation))
        print("Actual output : " + str(y[i]) + " Predicted Output␣
↪"+str(target_output))
        if(target_output != y[i]):
            print(".........\nUpdating Weights")
            w[0] = w[0] + learning_rate*(y[i] - target_output)*a[i]
            w[1] = w[1] + learning_rate*(y[i] - target_output)*b[i]
            print("Updated Weights: ", str(w[0]) + ','+str(w[1]))
            i = -1
            print("\nWeights Updated Training Again : ")
            print("###################################")
        i = i+1
        print("---------------")
    summation = and_input[0]*w[0] + and_input[1]*w[1] + bias*w[2]
    import matplotlib.pyplot as plt
    plt.figure(figsize=(12, 8))
    x_min, x_max = -0.5,1.5
    y_min, y_max = -0.5,1.5
    xx,yy = np.meshgrid(np.linspace(x_min, x_max,100),np.linspace(y_min,␣
↪y_max,100))

    Z = np.array([activation((np.dot([w[0],w[1]], [a, b]) + bias*w[2]),␣
↪threshold) for a, b in np.c_[xx.ravel(), yy.ravel()]])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap='Oranges')
    for i in range(len(y)):
        if y[i] == 0:
            plt.scatter(a[i], b[i], color='red', marker='x', label='Class 0')
        else:
            plt.scatter(a[i], b[i], color='green', marker='o', label='Class 1')

    #plt.scatter(a,b, c=y, cmap = "Blues_r", label = "a & b values")
    plt.title("OR GATE ")
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    #plt.xticks(())
    #plt.yticks(())
    #plt.grid()
    plt.legend()
    plt.show()
    return activation(summation, threshold)
```

```
and_input = [1,1]
print("OR Gate Output for "+ str(and_input) + " : " +␣
↪str(perceptron(and_input)))
```

```
Perceptron Training
####################
----------------
Input : 0 , 0
Weights : 0.6 , 0.6
Summation : -1.0
Actual output : 0 Predicted Output 0
---------------
Input : 0 , 1
Weights : 0.6 , 0.6
Summation : -0.4
Actual output : 1 Predicted Output 0
…
Updating Weights
Updated Weights:  0.6,1.1

Weights Updated Training Again :
##################################
---------------
Input : 0 , 0
Weights : 0.6 , 1.1
Summation : -1.0
Actual output : 0 Predicted Output 0
---------------
Input : 0 , 1
Weights : 0.6 , 1.1
Summation : 0.10000000000000009
Actual output : 1 Predicted Output 1
---------------
Input : 1 , 0
Weights : 0.6 , 1.1
Summation : -0.4
Actual output : 1 Predicted Output 0
…
Updating Weights
Updated Weights:  1.1,1.1

Weights Updated Training Again :
##################################
---------------
Input : 0 , 0
Weights : 1.1 , 1.1
Summation : -1.0
```

```
Actual output : 0 Predicted Output 0
---------------
Input : 0 , 1
Weights : 1.1 , 1.1
Summation : 0.10000000000000009
Actual output : 1 Predicted Output 1
---------------
Input : 1 , 0
Weights : 1.1 , 1.1
Summation : 0.10000000000000009
Actual output : 1 Predicted Output 1
---------------
Input : 1 , 1
Weights : 1.1 , 1.1
Summation : 1.2000000000000002
Actual output : 1 Predicted Output 1
---------------
```




```
OR Gate Output for [1, 1] : 1
```

3 Multi Layer Perceptron for X-OR

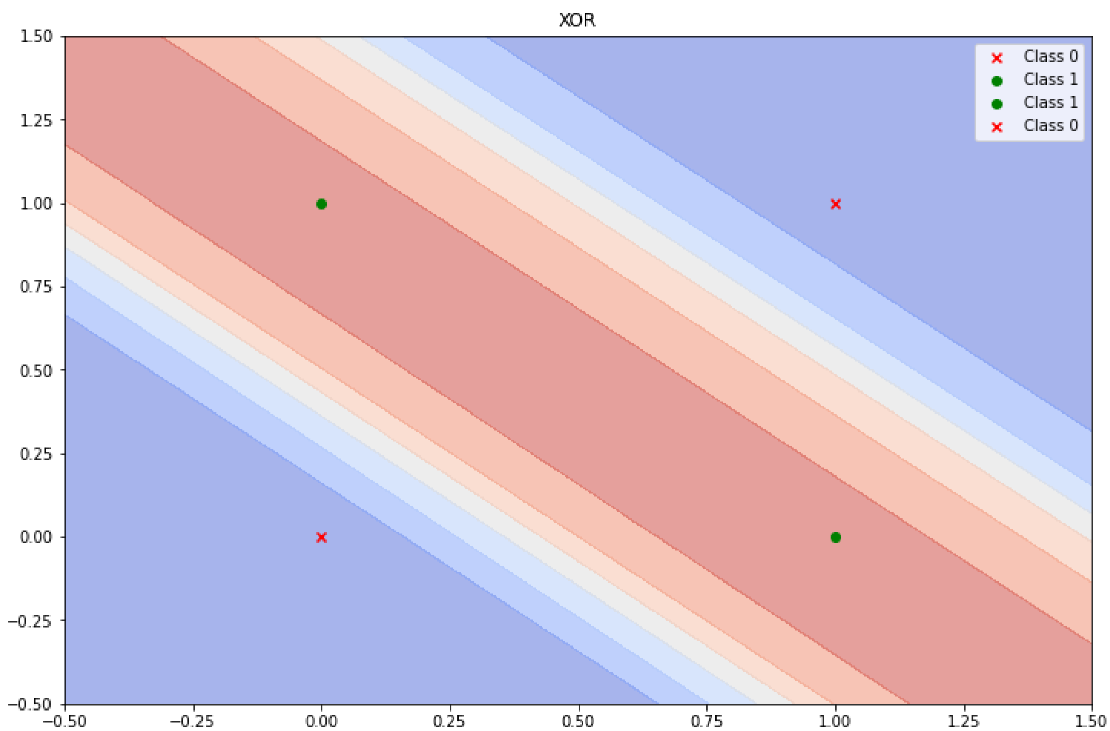
[3]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the input data
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

# Define the target output data
y = np.array([0, 1, 1, 0])

# Define the learning rate
learning_rate = 0.1

# Define the number of neurons in the hidden layer
hidden_layer_size = 2

# Define the initial weights and biases for the hidden layer
hidden_layer_weights = np.random.normal(size=(2, hidden_layer_size))
hidden_layer_bias = np.zeros(hidden_layer_size)

# Define the initial weights and bias for the output layer
output_layer_weights = np.random.normal(size=(hidden_layer_size, 1))
output_layer_bias = 0.0

# Define the activation function (sigmoid function)
def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))

# Define the derivative of the activation function (sigmoid function)
def sigmoid_derivative(x):
    return x * (1.0 - x)

# Train the perceptron
for i in range(10000):
    # Forward propagation
    hidden_layer_activation = sigmoid(np.dot(X, hidden_layer_weights) +␣
 ↪hidden_layer_bias)
    output_layer_activation = sigmoid(np.dot(hidden_layer_activation,␣
 ↪output_layer_weights) + output_layer_bias)

    # Backpropagation
    output_layer_error = y.reshape(-1, 1) - output_layer_activation
    output_layer_delta = output_layer_error *␣
 ↪sigmoid_derivative(output_layer_activation)

    hidden_layer_error = output_layer_delta.dot(output_layer_weights.T)
```

```
    hidden_layer_delta = hidden_layer_error *␣
↪sigmoid_derivative(hidden_layer_activation)

    # Update the weights and biases
    output_layer_weights += hidden_layer_activation.T.dot(output_layer_delta) *␣
↪learning_rate
    output_layer_bias += np.sum(output_layer_delta, axis=0, keepdims=True) *␣
↪learning_rate

    hidden_layer_weights += X.T.dot(hidden_layer_delta) * learning_rate
    hidden_layer_bias += np.sum(hidden_layer_delta, axis=0) * learning_rate


# Predictions
t= np.array([[1,1]])
hidden_layer_activation = sigmoid(np.dot(t, hidden_layer_weights) +␣
↪hidden_layer_bias)
output_layer_activation = sigmoid(np.dot(hidden_layer_activation,␣
↪output_layer_weights) + output_layer_bias)
a2 = np.squeeze(output_layer_activation)
if a2>=0.5:
    print("For input", t[0], "output is 1")
else:
    print("For input", t[0], "output is 0")
#print(output_layer_activation)
```

```
For input [1 1] output is 0
```

[4]:
```
# Plot the decision boundary
plt.figure(figsize=(12, 8))
x1 = np.linspace(-0.5, 1.5, 10)
x2 = np.linspace(-0.5, 1.5, 10)
X1, X2 = np.meshgrid(x1, x2)
Z = np.zeros_like(X1)

for i in range(X1.shape[0]):
    for j in range(X1.shape[1]):
        hidden_layer_activation = sigmoid(np.dot(np.array([X1[i,j], X2[i,j]]),␣
↪hidden_layer_weights)+ hidden_layer_bias)
        output_layer_activation = sigmoid(np.dot(hidden_layer_activation,␣
↪output_layer_weights)+ output_layer_bias)
        Z[i,j] = output_layer_activation[0]
plt.contourf(X1, X2, Z, cmap='coolwarm', alpha=0.5)
#plt.colorbar()
# Plot the input data
for i in range(len(X)):
    if y[i] == 0:
```

```
        plt.scatter(X[i][0], X[i][1], color='red', marker='x', label='Class 0')
    else:
        plt.scatter(X[i][0], X[i][1], color='green', marker='o', label='Class␣
↪1')

# Add labels and legend

plt.title('XOR')
plt.legend()
plt.show()
```



4 Information Gain with Decision Tree

[5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[16]:
```
data = pd.read_excel("/content/DataSet.xlsx")
data.head()
```

[16]:

	A	B	C	Output
0	1	1	5	S

```
1  0  1  1      L
2  1  2  5      S
3  0  2  1      L
4  0  1  1      M
```

[17]:
```
count_1 = 0
count_2 = 0
count_3 = 0
for i in range(len(data)):
    if(data["Output"][i]=="S"):
        count_1 = count_1+1
    elif(data["Output"][i]=="M"):
        count_2 = count_2 +1
    else:
        count_3 = count_3 +1
```

[18]:
```
print("Total S : ", count_1, "\nTotal M : ", count_2,"\nTotal L : ", count_3)
```

```
Total S :  10
Total M :  12
Total L :  8
```

[19]:
```
def cal_Entropy(x,y,w):
    t = x+y+w
    if(x == 0):
        entropy = -((y/t)*np.log2(y/t)+(w/t)*np.log2(w/t))
    elif(y == 0):
        entropy = -((x/t)*np.log2(x/t)+(w/t)*np.log2(w/t))
    elif(w == 0):
        entropy = -((x/t)*np.log2(x/t)+(y/t)*np.log2(y/t))
    else:
        entropy = -((x/t)*np.log2(x/t)+(y/t)*np.log2(y/t)+(w/t)*np.log2(w/t))
    #entropy  = "{:.2f}".format(entropy)
    #t = "{:.2f}".format(t)
    return t, entropy
```

[20]:
```
total_data, parent_Entropy = cal_Entropy(count_1,count_2,count_3)

print(f"Total Data : {total_data} \nParent Entropy : {parent_Entropy}")
```

```
Total Data : 30
Parent Entropy : 1.5655962303576019
```

[21]:
```
def child_data(c_data, x):
    child1= []
    child2= []
    #print(c_data)
```

```
    for i in range(len(c_data)):
        if(c_data[x][i]==1):
            child1.append(c_data["Output"][i])
        else:
            child2.append(c_data["Output"][i])
    #print(len(child1), len(child2))
    #child = pd.DataFrame(list(zip(child1, child2)),columns =['child1',␣
↪'child2'])
    return child1, child2

def count_data(data):
    count_1 = 0
    count_2 = 0
    count_3 = 0
    for i in range(len(data)):
        if(data[i]=="S"):
            count_1 = count_1+1
        elif(data[i]=="M"):
            count_2 = count_2 +1
        else:
            count_3 = count_3 +1
    return count_1, count_2, count_3

def wgh_avg_entropy(total_data, child_total_data1, child_entropy1,␣
↪child_total_data2, child_entropy2):
    avg_entropy = (((child_total_data1 / total_data)*child_entropy1) +␣
↪((child_total_data2 / total_data)*child_entropy2))
    return avg_entropy

def information_gain(child1, child2):
    count_1, count_2, count_3 = count_data(child1)
    child_total_data1, child_entropy1 = cal_Entropy(count_1,count_2,count_3)
    count_1, count_2, count_3 = count_data(child2)
    child_total_data2, child_entropy2 = cal_Entropy(count_1,count_2,count_3)
    avg_entropy = wgh_avg_entropy(total_data, child_total_data1,␣
↪child_entropy1, child_total_data2, child_entropy2)
    I_Gain= "{:.2f}".format(parent_Entropy-avg_entropy)
    return I_Gain
```

[22]:
```
child1, child2 = child_data(data[["A", "Output"]], "A")
print("Information Gain for attribute A : ", information_gain(child1, child2))
```

```
Information Gain for attribute A :  0.04
```

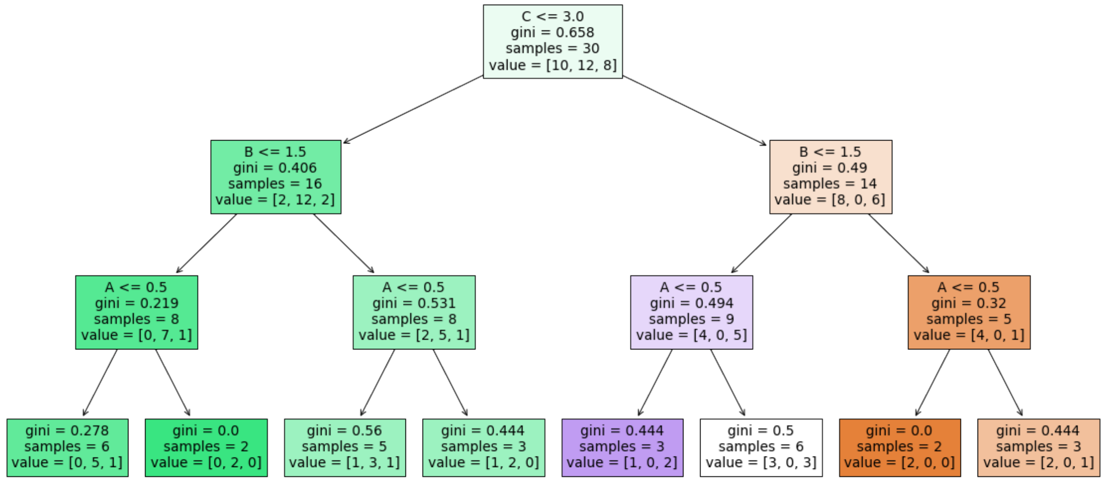
[23]:
```
child1, child2 = child_data(data[["B", "Output"]], "B")
print("Information Gain for attribute B : ", information_gain(child1, child2))
```

```
Information Gain for attribute B :  0.06
```

[24]:
```
child1, child2 = child_data(data[["C", "Output"]], "C")
print("Information Gain for attribute C : ", information_gain(child1, child2))
```

```
Information Gain for attribute C :  0.54
```

[25]:
```
import sys
import matplotlib
matplotlib.use('Agg')
%matplotlib inline
import pandas
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt

df = pandas.read_excel("DataSet.xlsx")

d = {'S': 0, 'M': 1, 'L': 2}
df["Output"] = df['Output'].map(d)

features = ['A', 'B', 'C']

X = df[features]
y = df['Output']

dtree = DecisionTreeClassifier(max_depth = 3)
dtree = dtree.fit(X, y)
plt.figure(figsize=(20, 10))
tree.plot_tree(dtree, feature_names=features,filled = True)
#plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```


C <= 3.0
gini = 0.658
samples = 30
value = [10, 12, 8]
B <= 1.5
gini = 0.406
samples = 16
value = [2, 12, 2]
B <= 1.5
gini = 0.49
samples = 14
value = [8, 0, 6]
A <= 0.5
gini = 0.219
samples = 8
value = [0, 7, 1]
A <= 0.5
gini = 0.531
samples = 8
value = [2, 5, 1]
A <= 0.5
gini = 0.494
samples = 9
value = [4, 0, 5]
A <= 0.5
gini = 0.32
samples = 5
value = [4, 0, 1]
gini = 0.278
samples = 6
value = [0, 5, 1]
gini = 0.0
samples = 2
value = [0, 2, 0]
gini = 0.56
samples = 5
value = [1, 3, 1]
gini = 0.444
samples = 3
value = [1, 2, 0]
gini = 0.444
samples = 3
value = [1, 0, 2]
gini = 0.5
samples = 6
value = [3, 0, 3]
gini = 0.0
samples = 2
value = [2, 0, 0]
gini = 0.444
samples = 3
value = [2, 0, 1]

```
[ ]:
```