```python
# implementation_from_scratch.py
# Main file for the implementation of the C4.5 algorithm
import pandas as pd
import numpy as np
import graphviz
from graphviz import Digraph
from math import log2
from copy import deepcopy
from weka_implementation import build_weka_tree
import PySimpleGUI as sg
from PIL import Image, ImageTk, ImageSequence
from multiprocessing import Process, Queue
import multiprocessing as mp
import time
import os.path
from os import path
import PIL.Image
import io
import base64
import math


# Louise and Aideen
def main():

    # Get the data, the train/test split percentage and the filepath of the data location
    data, split = gather_data()

    # While animate a 'loading' gif to show that the process is running,
    # Build a C4.5 tree from the data.
    # Return the root node of the tree, the dataset to use in testing and the time it took to build the tree
    root_node, testing_data, python_time_to_build = createTreeWhileShowingLoadingWindow(data, split)

    # Draw the tree and store it in png format
    print_tree(root_node)

    # Calculate the accuracy of the built tree using the testing data
    python_accuracy = test_tree(root_node, testing_data, split)

    # Build a tree using the same data in weka.
    # Draw the weka tree and store it in png format
    # Get the accuracy of the weka tree, and the time it took to construct
    weka_accuracy, weka_time_to_build = build_weka_tree(split)

    # Scale and format the times it took to run each implementation
    python_time = processTimes(python_time_to_build)
    weka_time = processTimes(weka_time_to_build)

    # Display both trees side to side, with their accuracies, and the time it took to build them
    DisplayTreesPopup(python_accuracy, weka_accuracy, python_time, weka_time)


# Aideen McLoughlin - 17346123
# Take in the time it took to run a process in seconds
# and return a nicely scaled representation of that value
def processTimes(time):
    if time < 1:
        scaledtime = round(time * 1000)
        timemsg = str(scaledtime) + " milliseconds"
    elif time <= 60:
        timemsg = str(round(time)) + " seconds"
    else:
        minutes = math.floor(time/60)
        seconds = round(time - minutes*60)
        timemsg = str(minutes) + " minutes and "+str(seconds)+" seconds"
    return "The tree took " + timemsg + " to build"


# Aideen McLoughlin - 17346123
# Using python multiprocessing, build a tree while showing a 'loading' animation
def createTreeWhileShowingLoadingWindow(data, split):

    # Create an 'Event' to indicate when the tree is built
    quit = mp.Event()
    # Create a 'Queue' to store generated values in
    Q = Queue()

    # Define both processes in the multiprocessing - the tree creation and the loading animation
    p1 = Process(target = createTree, args=(data, split, quit, Q,))
    p2 = Process(target = renderLoadingWindow, args=(quit, ))

    # Store the time before starting to build the tree
    starttime = time.time()

    # Start both processes
    p1.start()
    p2.start()

    # Wait for the quit event to be set, whch will happen once the tree is built
    quit.wait()

    # Store the time once the tree has been built
    endtime = time.time()

    # Get the data stored in the Queue
    queue_data = Q.get()

    # Return the Queue data and the time to build
    return queue_data[0], queue_data[1], endtime-starttime


# louise Kilheeney -16100463
def createTree(data, data_split, quit, queue):

    #call function to split the data into training and test data.
    train_data, test_data = split_data_training_testing(data, (data_split))

    #list of attributes in the data
    attributes = ['calorific_value','nitrogen','turbidity','style','alcohol','sugars','bitterness','colour','degree_of_fermentation']

    #calling function to build tree with the traning data and list of attributes
    root_node = build_tree(train_data, attributes)

    queue.put([root_node, test_data])
    queue.cancel_join_thread()
    quit.set()
    return True


# Aideen McLoughlin - 17346123
def renderLoadingWindow(quit):
    # Declare the PySimpleGUI layout for the popup window
    layout_loading = [[sg.Text("Loading")],[sg.Image(r'loading.gif', key='-IMAGE-')]]

    # Create the popup window
    window = sg.Window("Building C4.5 Tree", layout_loading, element_justification='c', margins=(0,0), element_padding=(0,0), finalize=True)

    # Animate the loading gif for the duration of time that 'quit' is not set
    interframe_duration = Image.open(r'loading.gif').info['duration']
    while not quit.is_set():
        event, values = window.read(timeout=interframe_duration)
        if event == sg.WIN_CLOSED:
            exit()
            break

        window.FindElement("-IMAGE-").UpdateAnimation("loading.gif",time_between_frames=interframe_duration)

    # Close the popup window
    window.close()
    return True


# Aideen McLoughlin - 17346123
def getInputData():
    # Declare the PySimpleGUI layout for the popup window
    layout = [
        [sg.Text('Data file: '), sg.InputText("beer.csv", key="file")],
        [sg.Text("Select Train/Test Data Split")],
        [sg.Radio('1/3',"1", key="1/3"),
           sg.Radio('1/2',"1", key="1/2"),
           sg.Radio('2/3',"1", key="2/3", default=True)],
         [sg.Button('Ok'), sg.Button('Quit')]]


    # Create the popup window
    window = sg.Window("Select Train/Test Split", layout)

    # Loop until the window is closed with 'x' 'OK' or 'Quit'
    # Set split to be the selected train/test split value
    # And set the filepath to be the contents of the InputText box (default beer.csv)
    while True:
        event, values = window.read()
        if event == sg.WIN_CLOSED or event == 'Quit':
            break
        if event == 'Ok':
            if values["1/3"] == True:
                split = (1/3)
            elif values["1/2"] == True:
                split = (1/2)
            elif values["2/3"] == True:
                split = (2/3)
            break
    filepath = values["file"]

    # Close the popup window
    window.close()
    return split, filepath


# Aideen McLoughlin - 17346123
def errorWindow(text):
    # Declare the PySimpleGUI layout for the popup window
    layout = [[sg.Text(text)],[sg.Button('Ok')]]

    # Create the popup window
    window = sg.Window("Error", layout)

    # Display a popup window with the text passed as a function param, until the user closes the window
    while True:
        event, values = window.read()
        if event == sg.WIN_CLOSED or event == 'Ok':
            break
```

```python

    # Close the popup window
    window.close()


# Aideen McLoughlin - 17346123
# Resixe the tree png images to be equal and fit nicely into the popup window
def resize_images():
    for image in (r'weka-test.gv.png', r'test.gv.png'):
        img = PIL.Image.open(image)
        img = img.resize((400, 400), PIL.Image.ANTIALIAS)
        img.save(image, format="PNG")


# Aideen McLoughlin - 17346123
def DisplayTreesPopup(python_accuracy, weka_accuracy, p_time, w_time):

    # Resize the tree images to be the right size for the popup
    resize_images()

    # Define 2 columns for the layout
    # The right one for the custome python implementation from scratch
    # The left one for the implementation with weka
    weka_column = [
        [sg.Text("Weka Implementation",font=('Helvetica 20'))],
        [sg.Image(r'weka-test.gv.png',key='-IMAGE-')],
        [sg.Text("Accuracy: "+str(weka_accuracy)+"%")],
        [sg.Text(w_time)]
    ]
    python_column = [
        [sg.Text("Our Python Implementation",font=('Helvetica 20'))],
        [sg.Image(r'test.gv.png',key='-IMAGE-')],
        [sg.Text("Accuracy: "+str(python_accuracy*100)+"%")],
        [sg.Text(p_time)]
    ]

    # Declare the PySimpleGUI layout for the popup window with the two columns, and a QUIT button
    layout = [
        [
            sg.Column(weka_column),
            sg.Column(python_column)
        ],
        [sg.Button('Quit')],
    ]

    # Create the popup window
    window = sg.Window("Generated C4.5 Tree", layout)

    # Display the popup window  until the user closes it
    while True:
        event, values = window.read()
        if event == sg.WIN_CLOSED or event == 'Quit': # if user closes window or clicks cancel
            break

    # Close the popup window
    window.close()


# louise Kilheeney - 16100463
def build_tree(data, attributes):
    #1. check the base cases
        #•  All the examples from the training set belong to the same class ( a tree leaf labeled with that class is returned ).
    if data['style'].nunique() == 1:
         return Node(True, data['style'][0], None)
    #•  The training set is empty ( returns a tree leaf called failure ).
    if len(data) == 0:
        return Node(True, "Fail", None)

    #2. find attribute with highest info gain, retrun best_attribute
        # calling function find-best-attribute which retruns the best attribute, the attribute subsets and the threshold divisor
    best_attribute, attribute_subsets, threshold_divisor  = find_best_attribute(data, attributes)

    #if best attribute is empty
    if best_attribute == "":
        #calling function get majorityclass to return the majority class
        majClass = getMajorityClass(data)
        return Node(True, majClass, None)
    else:
        #3. split the set (data) in subsets arrcording to value of best_attribute
        #attribute_subsets = split_into_subsets(best_attribute, data)
        remainColumns = deepcopy(data)
        remainColumns = data.drop(columns=[best_attribute])

        #4. repeat steps for each subset
        node = Node(False, best_attribute, threshold_divisor)
        for attr_subset in attribute_subsets:
            node.children.append(build_tree(attr_subset, remainColumns))

        return node



# Louise Kilheeney - 16100463
# Generate a png of the tree from the root node
def print_tree(root_node):

    # Create a diagraph in which to store the tree data
    g = Digraph('python_tree_implementation')

    # Add the root node, and all its children recursively
    addEl(root_node, g, 'a')

    # Format the graph as a png, and save it
    g.format = "png"
    g.render('test.gv', view=False)

# Louise Kilheeney - 16100463
# Add a node to the tree
def addEl(node, g, rootname):

    # If the node is not a leaf
    if not node.isLeaf:
        #Create the node
        g.node(name=str(rootname), label=node.label)

        # Create an edge from the node to its left child
        nodename1 = rootname+'b'
        g.edge(rootname, nodename1, label="<= "+str(round(node.divisor,2)))
        # Recursively add the nodes left child
        addEl(node.children[0],g,nodename1)

        # Create an edge from the node to its right child
        nodenamec = rootname + 'c'
        g.edge(rootname, nodenamec, label="> "+str(round(node.divisor,2)))
        # Recursively add the nodes right child
        addEl(node.children[1],g,nodenamec)
    else:
        # If the node is a leaf, add it while styling it as a leaf
        g.node(name=rootname, label=node.label, shape='box', style='filled')
```

```

# Louise Kilheeney - 16100463
def find_best_attribute(train_data, attributes):
    #  Returns the best attribute from all
    best_information_gain = 0
    best_attribute = ""
    threshold_divisor = ""
    subsets = []
    for attribute in attributes:
        #making sure not to include style
        if attribute != 'style':

            #calling function split_into_subsets
            temp_subsets, temp_divisor = split_into_subsets(attribute, train_data)

            # temp gain is equal to the information gain function for the train_data
            and the subsets.
            temp_gain = information_gain(train_data, temp_subsets)

            #check for the best attribute
            if temp_gain > best_information_gain:
                best_attribute = attribute
                best_information_gain = temp_gain
                subsets = temp_subsets
                threshold_divisor = temp_divisor
    # return the best attribute , subsets and the threshold divisor
    return best_attribute, subsets, threshold_divisor


# Louise Kilheeney - 16100463
# Function to get the majority class of the data been passed in.
def getMajorityClass(data):
    #find the majority class in the data with the data - style
    grouped = data.groupby(data['style'])
    #return majority class
    return max(grouped.groups)


# Louise Kilheeney - 16100463
def split_into_subsets(column_header, training_data):
    split_values = []
    maxEnt = -1*float("inf")
    best_threshold = ""
    sorted_data = training_data.sort_values(by=[column_header])
    for item in range(0, len(training_data) - 1):
        if type(sorted_data.iloc[item][column_header]) != 'style':
            if sorted_data.iloc[item][column_header] !=
            sorted_data.iloc[item+1][column_header]:
                threshold = (sorted_data.iloc[item][column_header] +
                sorted_data.iloc[item+1][column_header]) / 2
                smaller_than_threshold = pd.DataFrame()
                bigger_than_threshold = pd.DataFrame()
                for index, row in sorted_data.iterrows():
                    if(row[column_header] > threshold):
                        bigger_than_threshold = bigger_than_threshold.append(row,
                        ignore_index = True)
                    else:
                        smaller_than_threshold = smaller_than_threshold.append(row,
                        ignore_index = True)

                igain = information_gain(training_data, [smaller_than_threshold,
                bigger_than_threshold])

                if igain >= maxEnt:
                    split_values = [smaller_than_threshold, bigger_than_threshold]
                    best_threshold = threshold
                    maxEnt = igain
```

```python
    return split_values, best_threshold


# Aideen McLoughlin - 17346123
# Split the python DataFrame object into 3 dataFrame objects
# One storing all the values with the syle 'ale'
# One storing all the values with the syle 'lager'
# and One storing all the values with the syle 'stout'
def split_data_styles(data):

    # Declare empty subsets array
    subsets = {}

    #Group the data passed to the function by its style
    grouped = data.groupby(data['style'])

    # For each style name, add the values in that group to the subsets array as a new Dataframe object
    for index, beer_style in enumerate(['ale','lager','stout']):
        if beer_style in grouped.groups.keys():
            subsets[index] = grouped.get_group(beer_style)
        else:
            subsets[index] = {}

    # return the subsets array of DataFrame objects
    return subsets


# Louise Kilheeney - 16100463
# Split the data into training and testing datasets
def split_data_training_testing(data, ratio):

    # Drop the beer_id column as it is not relevant to the beer style
    data = data.drop(columns=['beer_id'])

    # Get a random sample from the data file as the training data
    # This data will be ratio% of the initial dataset
    train = data.sample(frac=ratio,random_state=5)

    # Get the rest of the dataset values as the testing data
    test = data.merge(train, how='left', indicator=True)
    test = test[(test['_merge']=='left_only')].copy()
    test = test.drop(columns='_merge').copy()

    # Save the divided dataset so that it can be used in the weka implementation
    train.to_csv('train_data_generated.csv',index=False,header=True)
    test.to_csv('test_data_generated.csv',index=False,header=True)

    # Create a folder to store the reults in if it does not already exist
    if not path.exists('results'):
        print("Create results folder")
        os.mkdir('results')

    # Return the training and testing data
    return train, test

from pandas.errors import EmptyDataError
# Louise Kilheeney - 16100463
# Get the filepath of the data file, and the train/test data split fro user imput in a PySimpleGUI popup
# If a filepath provided is not valid, prompt the user to input a new filepath.
# Repeat until a valid filepath is provided
def gather_data():

    # Get the train/test split and the filepath of the data file
    split, filepath = getInputData()

    # Create an empty pandas dataframe element
    data = pd.DataFrame()

    # While the dataframe element remains empty
    while data.empty:

        # Check If the filepath is valid
        if path.isfile(filepath):
            # If it is, set the data to be the csv data at that filepath
            data = pd.read_csv(filepath)
        else:
            # If it is not valid, Display an error pop-up and prompt the user to imput the split and filepath again
            errorWindow("File not found, please try again")
            split, filepath = getInputData()

    # Once the dataframe element is filled, return the data, the train/test split percentage and the filepath (For use in the weka implementation)
    return data, split


# Aideen McLoughlin - 17346123
# Calculate the entropy of the passed data set
def entropy(dataset):

    # Initialise entropy to zero value
    entropy = 0

    # Get the ale, lager and stout subset DataFrames from the passed dataset
    subsets = split_data_styles(dataset)

    # For each subset
    for index in range(len(subsets)):

        # Get the percentage of the dataset which is in the subset
        probability = len(subsets[index])/ len(dataset)

        # If the probability is not zero,
        # Subtract plog2(p) from the entropy value where p is probability
        if probability != 0:
            entropy = entropy - (probability)*log2(probability)

    # Return the entropy value
    return entropy


# Louise Kilheeney - 16100463
# function to calculate the information gain
def information_gain(train_target, subsets):
    #getting the entropy value of  the train_target
    entropyTarget = entropy(train_target)
    total = len(train_target)

    Gain = entropyTarget

    #for each subset
    for i in range(0, len(subsets)):
        #length of each subset
        numBeer = len(subsets[i])
        #Gain = Ent(S) - |S beer =ale |/|S|*( Ent( S beer=ale )) -  |S beer=stout |/|S|*( Ent( S beer=stout )) - |S beer=lager |/|S|*( Ent( S beer=lager )) - ....
        firstPart = numBeer/total
        secondPart = entropy(subsets[i])
        Gain -= (firstPart*secondPart)
    # Return the information gain value
    return Gain
```

```python

# Aideen McLoughlin - 17346123
# Test the built tree using the testing data
def test_tree(root_node, testing_data, split):

    # Get the style as the target values, and then drop them from the testing dataset
    test_target = testing_data['style'].values
    testing_data = testing_data.drop(columns=['style'])

    # get the results of the tree predictions for all the testing data values
    test_results = test_data(testing_data, root_node, [])

    # Initialise the number of correct entries to 0
    correct = 0

    accurate = []
    # For each test result, check if it is accurate using the style values we removed
    from the Dataframe earlier
    # Keep a count of the number of correct predictions
    # If the prediction is wrong, print the incorrect predicton
    for index in range(0, len(test_results)):
        if test_results[index] == test_target[index]:
            correct = correct +1
            accurate.append(1)
        else:
            accurate.append(0)

    # Calculate the accuracy to 2 decimal places, and return it
    accuracy = round(correct/len(test_results),2)

    df = pd.DataFrame()
    df['Actual'] = test_target
    df['Predicted'] = test_results
    df['Accuracy'] = accurate

    filename =
    "results/python-results-"+str(round(split,2))+"-"+str(round(accuracy,2))+".csv"
    df.to_csv(filename,index=False,header=True)

    return accuracy


# Aideen McLoughlin - 17346123
# Using the root node of the constructed tree, predict the output of all the test data
inputs
def test_data(data, node, test_results):

    # For each data value, get the predicted result
    for item in range(0, len(data)):
        test_results.append(test_lr(node, data.iloc[item]))

    # return the set of all predicted results
    return test_results


# Aideen McLoughlin - 17346123
# Get the final leaf node destination for a data row
def test_lr(node, row):

    # Decide which child path of a node to proceed into, based on the input value.
    # This function will call itself recursively until it reaches a leaf node
    # That leaf node will be returned to the function which called it
    if node.isLeaf:
        return node.label
    else:
        if row[node.label] <= node.divisor:
            return test_lr(node.children[0], row)
        else:
            return test_lr(node.children[1], row)


# Louise Kilheeney - 16100463
# Node class
class Node:
    def __init__(self,isLeaf, label, divisor):
        self.label = label
        self.isLeaf = isLeaf
        self.divisor = divisor
        self.children = []


# Louise Kilheeney - 16100463
# Set the main function to run when the file is run
if __name__ == '__main__':
    main()

```