```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 Kilheeney -16100463
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)
```

```
65          timemsg = str(minutes) + " minutes and "+str(seconds)+" seconds"
66      return "The tree took " + timemsg + " to build"
67
68
69  # Aideen McLoughlin - 17346123
70  # Using python multiprocessing, build a tree while showing a 'loading' animation
71  def createTreeWhileShowingLoadingWindow(data, split):
72
73      # Create an 'Event' to indicate when the tree is built
74      quit = mp.Event()
75      # Create a 'Queue' to store generated values in
76      Q = Queue()
77
78      # Define both processes in the multiprocessing - the tree creation and the loading
        animation
79      p1 = Process(target = createTree, args=(data, split, quit, Q,))
80      p2 = Process(target = renderLoadingWindow, args=(quit, ))
81
82      # Store the time before starting to build the tree
83      starttime = time.time()
84
85      # Start both processes
86      p1.start()
87      p2.start()
88
89      # Wait for the quit event to be set, whch will happen once the tree is built
90      quit.wait()
91
92      # Store the time once the tree has been built
93      endtime = time.time()
94
95      # Get the data stored in the Queue
96      queue_data = Q.get()
97
98      # Return the Queue data and the time to build
99      return queue_data[0], queue_data[1], endtime-starttime
100
101
102 # louise Kilheeney -16100463
103 def createTree(data, data_split, quit, queue):
104
105     #call function to split the data into training and test data.
106     train_data, test_data = split_data_training_testing(data, (data_split))
107
108     #list of attributes in the data
109     attributes =
        ['calorific_value','nitrogen','turbidity','style','alcohol','sugars','bitterness','co
        lour','degree_of_fermentation']
110
111     #calling function to build tree with the traning data and list of attributes
112     root_node = build_tree(train_data, attributes)
113
114     queue.put([root_node, test_data])
115     queue.cancel_join_thread()
116     quit.set()
117     return True
118
119
120 # Aideen McLoughlin - 17346123
121 def renderLoadingWindow(quit):
122     # Declare the PySimpleGUI layout for the popup window
123     layout_loading = [[sg.Text("Loading")],[sg.Image(r'loading.gif', key='-IMAGE-')]]
124
125     # Create the popup window
126     window = sg.Window("Building C4.5 Tree", layout_loading, element_justification='c',
        margins=(0,0), element_padding=(0,0), finalize=True)
127
```

```
128         # Animate the loading gif for the duration of time that 'quit' is not set
129         interframe_duration = Image.open(r'loading.gif').info['duration']
130         while not quit.is_set():
131             event, values = window.read(timeout=interframe_duration)
132             if event == sg.WIN_CLOSED:
133                 exit()
134                 break
135 
            window.FindElement("-IMAGE-").UpdateAnimation("loading.gif",time_between_frames=i
            nterframe_duration)
136 
137         # Close the popup window
138         window.close()
139         return True
140 
141 
142     # Aideen McLoughlin - 17346123
143     def getInputData():
144         # Declare the PySimpleGUI layout for the popup window
145         layout = [
146             [sg.Text('Data file: '), sg.InputText("beer.csv", key="file")],
147             [sg.Text("Select Train/Test Data Split")],
148             [sg.Radio('1/3',"1", key="1/3"),
149                sg.Radio('1/2',"1", key="1/2"),
150                sg.Radio('2/3',"1", key="2/3", default=True)],
151              [sg.Button('Ok'), sg.Button('Quit')]]
152 
153 
154         # Create the popup window
155         window = sg.Window("Select Train/Test Split", layout)
156 
157         # Loop until the window is closed with 'x' 'OK' or 'Quit'
158         # Set split to be the selected train/test split value
159         # And set the filepath to be the contents of the InputText box (default beer.csv)
160         while True:
161             event, values = window.read()
162             if event == sg.WIN_CLOSED or event == 'Quit':
163                 break
164             if event == 'Ok':
165                 if values["1/3"] == True:
166                     split = (1/3)
167                 elif values["1/2"] == True:
168                     split = (1/2)
169                 elif values["2/3"] == True:
170                     split = (2/3)
171                 break
172         filepath = values["file"]
173 
174         # Close the popup window
175         window.close()
176         return split, filepath
177 
178 
179     # Aideen McLoughlin - 17346123
180     def errorWindow(text):
181         # Declare the PySimpleGUI layout for the popup window
182         layout = [[sg.Text(text)],[sg.Button('Ok')]]
183 
184         # Create the popup window
185         window = sg.Window("Error", layout)
186 
187         # Display a popup window with the text passed as a function param, until the user
            closes the window
188         while True:
189             event, values = window.read()
190             if event == sg.WIN_CLOSED or event == 'Ok':
191                 break
```

```
192
193         # Close the popup window
194         window.close()
195
196
197     # Aideen McLoughlin - 17346123
198     # Resixe the tree png images to be equal and fit nicely into the popup window
199     def resize_images():
200         for image in (r'weka-test.gv.png', r'test.gv.png'):
201             img = PIL.Image.open(image)
202             img = img.resize((400, 400), PIL.Image.ANTIALIAS)
203             img.save(image, format="PNG")
204
205
206     # Aideen McLoughlin - 17346123
207     def DisplayTreesPopup(python_accuracy, weka_accuracy, p_time, w_time):
208
209         # Resize the tree images to be the right size for the popup
210         resize_images()
211
212         # Define 2 columns for the layout
213         # The right one for the custome python implementation from scratch
214         # The left one for the implementation with weka
215         weka_column = [
216             [sg.Text("Weka Implementation",font=('Helvetica 20'))],
217             [sg.Image(r'weka-test.gv.png',key='-IMAGE-')],
218             [sg.Text("Accuracy: "+str(weka_accuracy)+"%")],
219             [sg.Text(w_time)]
220         ]
221         python_column = [
222             [sg.Text("Our Python Implementation",font=('Helvetica 20'))],
223             [sg.Image(r'test.gv.png',key='-IMAGE-')],
224             [sg.Text("Accuracy: "+str(python_accuracy*100)+"%")],
225             [sg.Text(p_time)]
226         ]
227
228         # Declare the PySimpleGUI layout for the popup window with the two columns, and a
            QUIT button
229         layout = [
230             [
231                 sg.Column(weka_column),
232                 sg.Column(python_column)
233             ],
234             [sg.Button('Quit')],
235         ]
236
237         # Create the popup window
238         window = sg.Window("Generated C4.5 Tree", layout)
239
240         # Display the popup window  until the user closes it
241         while True:
242             event, values = window.read()
243             if event == sg.WIN_CLOSED or event == 'Quit': # if user closes window or clicks
                cancel
244                 break
245
246         # Close the popup window
247         window.close()
248
249
250     # louise Kilheeney - 16100463
251     def build_tree(data, attributes):
252         #1. check the base cases
253             #•  All the examples from the training set belong to the same class ( a tree
                leaf labeled with that class is returned ).
254         if data['style'].nunique() == 1:
255              return Node(True, data['style'][0], None)
```

```
256         #•  The training set is empty ( returns a tree leaf called failure ).
257         if len(data) == 0:
258             return Node(True, "Fail", None)
259 
260         #2. find attribute with highest info gain, retrun best_attribute
261             # calling function find-best-attribute which retruns the best attribute, the
                attribute subsets and the threshold divisor
262         best_attribute, attribute_subsets, threshold_divisor  = find_best_attribute(data,
            attributes)
263 
264         #if best attribute is empty
265         if best_attribute == "":
266             #calling function get majorityclass to return the majority class
267             majClass = getMajorityClass(data)
268             return Node(True, majClass, None)
269         else:
270             #3. split the set (data) in subsets arrcording to value of best_attribute
271             #attribute_subsets = split_into_subsets(best_attribute, data)
272             remainColumns = deepcopy(data)
273             remainColumns = data.drop(columns=[best_attribute])
274 
275             #4. repeat steps for each subset
276             node = Node(False, best_attribute, threshold_divisor)
277             for attr_subset in attribute_subsets:
278                 node.children.append(build_tree(attr_subset, remainColumns))
279 
280             return node
281 
282 
283     # Louise Kilheeney - 16100463
284     # Generate a png of the tree from the root node
285     def print_tree(root_node):
286 
287         # Create a diagraph in which to store the tree data
288         g = Digraph('python_tree_implementation')
289 
290         # Add the root node, and all its children recursively
291         addEl(root_node, g, 'a')
292 
293         # Format the graph as a png, and save it
294         g.format = "png"
295         g.render('test.gv', view=False)
296 
297 
298     # Louise Kilheeney - 16100463
299     # Add a node to the tree
300     def addEl(node, g, rootname):
301 
302         # If the node is not a leaf
303         if not node.isLeaf:
304             #Create the node
305             g.node(name=str(rootname), label=node.label)
306 
307             # Create an edge from the node to its left child
308             nodename1 = rootname+'b'
309             g.edge(rootname, nodename1, label="<= "+str(round(node.divisor,2)))
310             # Recursively add the nodes left child
311             addEl(node.children[0],g,nodename1)
312 
313             # Create an edge from the node to its right child
314             nodenamec = rootname + 'c'
315             g.edge(rootname, nodenamec, label="> "+str(round(node.divisor,2)))
316             # Recursively add the nodes right child
317             addEl(node.children[1],g,nodenamec)
318         else:
319             # If the node is a leaf, add it while styling it as a leaf
320             g.node(name=rootname, label=node.label, shape='box', style='filled')
```

```
321
322
323   # Louise Kilheeney - 16100463
324   def find_best_attribute(train_data, attributes):
325       #  Returns the best attribute from all
326       best_information_gain = 0
327       best_attribute = ""
328       threshold_divisor = ""
329       subsets = []
330       for attribute in attributes:
331           #making sure not to include style
332           if attribute != 'style':
333
334               #calling function split_into_subsets
335               temp_subsets, temp_divisor = split_into_subsets(attribute, train_data)
336
337               # temp gain is equal to the information gain function for the train_data
                  and the subsets.
338               temp_gain = information_gain(train_data, temp_subsets)
339
340               #check for the best attribute
341               if temp_gain > best_information_gain:
342                   best_attribute = attribute
343                   best_information_gain = temp_gain
344                   subsets = temp_subsets
345                   threshold_divisor = temp_divisor
346       # return the best attribute , subsets and the threshold divisor
347       return best_attribute, subsets, threshold_divisor
348
349
350   # Louise Kilheeney - 16100463
351   # Function to get the majority class of the data been passed in.
352   def getMajorityClass(data):
353       #find the majority class in the data with the data - style
354       grouped = data.groupby(data['style'])
355       #return majority class
356       return max(grouped.groups)
357
358
359   # Louise Kilheeney - 16100463
360   def split_into_subsets(column_header, training_data):
361       split_values = []
362       maxEnt = -1*float("inf")
363       best_threshold = ""
364       sorted_data = training_data.sort_values(by=[column_header])
365       for item in range(0, len(training_data) - 1):
366           if type(sorted_data.iloc[item][column_header]) != 'style':
367               if sorted_data.iloc[item][column_header] !=
                  sorted_data.iloc[item+1][column_header]:
368                   threshold = (sorted_data.iloc[item][column_header] +
                      sorted_data.iloc[item+1][column_header]) / 2
369                   smaller_than_threshold = pd.DataFrame()
370                   bigger_than_threshold = pd.DataFrame()
371                   for index, row in sorted_data.iterrows():
372                       if(row[column_header] > threshold):
373                           bigger_than_threshold = bigger_than_threshold.append(row,
                              ignore_index = True)
374                       else:
375                           smaller_than_threshold = smaller_than_threshold.append(row,
                              ignore_index = True)
376
377                   igain = information_gain(training_data, [smaller_than_threshold,
                      bigger_than_threshold])
378
379                   if igain >= maxEnt:
380                       split_values = [smaller_than_threshold, bigger_than_threshold]
381                       best_threshold = threshold
```

```
                    maxEnt = igain
    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


# Aideen McLoughlin - 17346123
# 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

# 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()

```

```
447         # Create an empty pandas dataframe element
448         data = pd.DataFrame()
449 
450         # While the dataframe element remains empty
451         while data.empty:
452 
453             # Check If the filepath is valid
454             if path.isfile(filepath):
455                 # If it is, set the data to be the csv data at that filepath
456                 data = pd.read_csv(filepath)
457             else:
458                 # If it is not valid, Display an error pop-up and prompt the user to imput
                    the split and filepath again
459                 errorWindow("File not found, please try again")
460                 split, filepath = getInputData()
461 
462         # Once the dataframe element is filled, return the data, the train/test split
            percentage and the filepath (For use in the weka implementation)
463         return data, split
464 
465 
466     # Aideen McLoughlin - 17346123
467     # Calculate the entropy of the passed data set
468     def entropy(dataset):
469 
470         # Initialise entropy to zero value
471         entropy = 0
472 
473         # Get the ale, lager and stout subset DataFrames from the passed dataset
474         subsets = split_data_styles(dataset)
475 
476         # For each subset
477         for index in range(len(subsets)):
478 
479             # Get the percentage of the dataset which is in the subset
480             probability = len(subsets[index])/ len(dataset)
481 
482             # If the probability is not zero,
483             # Subtract plog2(p) from the entropy value where p is probability
484             if probability != 0:
485                 entropy = entropy - (probability)*log2(probability)
486 
487         # Return the entropy value
488         return entropy
489 
490 
491     # Louise Kilheeney - 16100463
492     # function to calculate the information gain
493     def information_gain(train_target, subsets):
494         #getting the entropy value of  the train_target
495         entropyTarget = entropy(train_target)
496         total = len(train_target)
497 
498         Gain = entropyTarget
499 
500         #for each subset
501         for i in range(0, len(subsets)):
502             #length of each subset
503             numBeer = len(subsets[i])
504             #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 )) - ....
505             firstPart = numBeer/total
506             secondPart = entropy(subsets[i])
507             Gain -= (firstPart*secondPart)
508         # Return the information gain value
509         return Gain
510 
```

```
511
512 # Aideen McLoughlin - 17346123
513 # Test the built tree using the testing data
514 def test_tree(root_node, testing_data, split):
515
516     # Get the style as the target values, and then drop them from the testing dataset
517     test_target = testing_data['style'].values
518     testing_data = testing_data.drop(columns=['style'])
519
520     # get the results of the tree predictions for all the testing data values
521     test_results = test_data(testing_data, root_node, [])
522
523     # Initialise the number of correct entries to 0
524     correct = 0
525
526     accurate = []
527     # For each test result, check if it is accurate using the style values we removed
    from the Dataframe earlier
528     # Keep a count of the number of correct predictions
529     # If the prediction is wrong, print the incorrect predicton
530     for index in range(0, len(test_results)):
531         if test_results[index] == test_target[index]:
532             correct = correct +1
533             accurate.append(1)
534         else:
535             accurate.append(0)
536
537     # Calculate the accuracy to 2 decimal places, and return it
538     accuracy = round(correct/len(test_results),2)
539
540     df = pd.DataFrame()
541     df['Actual'] = test_target
542     df['Predicted'] = test_results
543     df['Accuracy'] = accurate
544
545     filename =
    "results/python-results-"+str(round(split,2))+"-"+str(round(accuracy,2))+".csv"
546     df.to_csv(filename,index=False,header=True)
547
548     return accuracy
549
550
551 # Aideen McLoughlin - 17346123
552 # Using the root node of the constructed tree, predict the output of all the test data
    inputs
553 def test_data(data, node, test_results):
554
555     # For each data value, get the predicted result
556     for item in range(0, len(data)):
557         test_results.append(test_lr(node, data.iloc[item]))
558
559     # return the set of all predicted results
560     return test_results
561
562
563 # Louise Kilheeney - 16100463
564 # Get the final leaf node destination for a data row
565 def test_lr(node, row):
566
567     # Decide which child path of a node to proceed into, based on the input value.
568     # This function will call itself recursively until it reaches a leaf node
569     # That leaf node will be returned to the function which called it
570     if node.isLeaf:
571         return node.label
572     else:
573         if row[node.label] <= node.divisor:
574             return test_lr(node.children[0], row)
```

```
575             else:
576                 return test_lr(node.children[1], row)
577 
578 
579     # Louise Kilheeney - 16100463
580     # Node class
581     class Node:
582         def __init__(self,isLeaf, label, divisor):
583             self.label = label
584             self.isLeaf = isLeaf
585             self.divisor = divisor
586             self.children = []
587 
588 
589     # Louise Kilheeney - 16100463
590     # Set the main function to run when the file is run
591     if __name__ == '__main__':
592         main()
593 
```