#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Mon Aug 26 14:41:21 2019

@author: soetcse

"""

# CART on the Bank Note dataset

from random import seed

from random import randrange

from csv import reader

# Load a CSV file

def load\_csv(filename):

file = open(filename, "r")

lines = reader(file)

dataset = list(lines)

return dataset

# Convert string column to float

def str\_column\_to\_float(dataset, column):

for row in dataset:

row[column] = float(row[column].strip())

# Split a dataset into k folds

def cross\_validation\_split(dataset, n\_folds):

dataset\_split = list()

dataset\_copy = list(dataset)

fold\_size = int(len(dataset) / n\_folds)

for i in range(n\_folds):

fold = list()

while len(fold) < fold\_size:

index = randrange(len(dataset\_copy))

fold.append(dataset\_copy.pop(index))

dataset\_split.append(fold)

return dataset\_split

# Calculate accuracy percentage

def accuracy\_metric(actual, predicted):

correct = 0

for i in range(len(actual)):

if actual[i] == predicted[i]:

correct += 1

return correct / float(len(actual)) \* 100.0

# Evaluate an algorithm using a cross validation split

def evaluate\_algorithm(dataset, algorithm, n\_folds, \*args):

folds = cross\_validation\_split(dataset, n\_folds)

scores = list()

for fold in folds:

train\_set = list(folds)

train\_set.remove(fold)

train\_set = sum(train\_set, [])

test\_set = list()

for row in fold:

row\_copy = list(row)

test\_set.append(row\_copy)

row\_copy[-1] = None

predicted = algorithm(train\_set, test\_set, \*args)

actual = [row[-1] for row in fold]

accuracy = accuracy\_metric(actual, predicted)

scores.append(accuracy)

return scores

# Split a dataset based on an attribute and an attribute value

def test\_split(index, value, dataset):

left, right = list(), list()

for row in dataset:

if row[index] < value:

left.append(row)

else:

right.append(row)

return left, right

# Calculate the Gini index for a split dataset

def gini\_index(groups, classes):

# count all samples at split point

n\_instances = float(sum([len(group) for group in groups]))

# sum weighted Gini index for each group

gini = 0.0

for group in groups:

size = float(len(group))

# avoid divide by zero

if size == 0:

continue

score = 0.0

# score the group based on the score for each class

for class\_val in classes:

p = [row[-1] for row in group].count(class\_val) / size

score += p \* p

# weight the group score by its relative size

gini += (1.0 - score) \* (size / n\_instances)

return gini

# Select the best split point for a dataset

def get\_split(dataset):

class\_values = list(set(row[-1] for row in dataset))

b\_index, b\_value, b\_score, b\_groups = 999, 999, 999, None

for index in range(len(dataset[0])-1):

for row in dataset:

groups = test\_split(index, row[index], dataset)

gini = gini\_index(groups, class\_values)

if gini < b\_score:

b\_index, b\_value, b\_score, b\_groups = index, row[index], gini, groups

return {'index':b\_index, 'value':b\_value, 'groups':b\_groups}

# Create a terminal node value

def to\_terminal(group):

outcomes = [row[-1] for row in group]

return max(set(outcomes), key=outcomes.count)

# Create child splits for a node or make terminal

def split(node, max\_depth, min\_size, depth):

left, right = node['groups']

del(node['groups'])

# check for a no split

if not left or not right:

node['left'] = node['right'] = to\_terminal(left + right)

return

# check for max depth

if depth >= max\_depth:

node['left'], node['right'] = to\_terminal(left), to\_terminal(right)

return

# process left child

if len(left) <= min\_size:

node['left'] = to\_terminal(left)

else:

node['left'] = get\_split(left)

split(node['left'], max\_depth, min\_size, depth+1)

# process right child

if len(right) <= min\_size:

node['right'] = to\_terminal(right)

else:

node['right'] = get\_split(right)

split(node['right'], max\_depth, min\_size, depth+1)

# Build a decision tree

def build\_tree(train, max\_depth, min\_size):

root = get\_split(train)

split(root, max\_depth, min\_size, 1)

return root

# Make a prediction with a decision tree

def predict(node, row):

if row[node['index']] < node['value']:

if isinstance(node['left'], dict):

return predict(node['left'], row)

else:

return node['left']

else:

if isinstance(node['right'], dict):

return predict(node['right'], row)

else:

return node['right']

# Classification and Regression Tree Algorithm

def decision\_tree(train, test, max\_depth, min\_size):

tree = build\_tree(train, max\_depth, min\_size)

predictions = list()

for row in test:

prediction = predict(tree, row)

predictions.append(prediction)

return(predictions)

# Test CART on Bank Note dataset

seed(1)

# load and prepare data

filename = 'data\_banknote\_authentication.csv'

dataset = load\_csv('/home/soetcse/Downloads/data\_banknote\_authentication.csv')

# convert string attributes to integers

for i in range(len(dataset[0])):

str\_column\_to\_float(dataset, i)

# evaluate algorithm

n\_folds = 5

max\_depth = 5

min\_size = 10

scores = evaluate\_algorithm(dataset, decision\_tree, n\_folds, max\_depth, min\_size)

print('Scores: %s' % scores)

print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))