```
# -*- coding: utf-8 -*-
"""19BCP093-Assgn-2-Decision-Tree.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1LmqUjMpoV4J5IMCF41AXj_qU6q
DbmiYk
"""

import numpy as np #math library
import pandas as pd #data analysis (Working on data)
import seaborn as sns #importing files dataset (graph plot also)
import matplotlib.pyplot as plt #graph plot
import random #random number generator

iris = sns.load_dataset("iris") #lodaing iris into variable iris
print(iris)

def train_test_split(dataframe, test_size): #randomly splitting into tr
ain and test variable 2D arrays

  indices = iris.index.tolist()
  #print(indices)
  test_indices = random.sample(population = indices, k = test_size) #ra
ndomly assigning index of some data rows from iris to our variable
  #print(test_indices)
  test_df = iris.loc[test_indices] #dataframe corresponding to test_ind
ices
  train_df = iris.drop(test_indices) #dataframe with left overs
  return train_df, test_df

train_df, test_df = train_test_split(iris, 30)

def check_purity(data): #yes no checker for purity

  label_column = data[: , -1] #making array of species column
  #print(label_column)
  unique_classes = np.unique(label_column) #making array of unique elem
ents(species) in label_column
  if(len(unique_classes) == 1):
    return True #pure
  return False #impure

def classify_data(data): #classify into sub class , basically returns t
he most similar species match from dataset when compared to entered num
ber entry data
```

```
  label_column = data[:, -1]
  unique_classes, count_unique_classes = np.unique(label_column, return
_counts = True)#making array of unique elements(species) in label_colum
n / #making array of how many same data entry(same species) of label co
lumns in data
  #print(unique_classes)
  #print(count_unique_classes)
  classification = unique_classes[count_unique_classes.argmax()]
  return classification

classify_data(test_df.values)

def get_potential_splits(data): #gives us from where we can split our d
ata for each column

  potential_splits = {} #dictionary
  _, n_columns = data.shape #shape returns number of rows and number of
 columns
  for column_index in range (n_columns -
1): #4 times loop runs dont need species
    potential_splits[column_index] = [] #make empty array in dictionary
    values = data[:,column_index] #value has our current column from da
ta
    #print(values)
    unique_values = np.unique(values) #no point in splitting data from
same values twice
    #print(unique_values)
    for index in range(len(unique_values)): # now calculate potenial sp
lits
        if index != 0:
            current_value = unique_values[index]
            previous_value = unique_values[index - 1]
            potential_split = (current_value + previous_value) / 2 #
            potential_splits[column_index].append(potential_split)
  return potential_splits

#print(get_potential_splits(test_df.values))

def split_data(data, split_column, split_value): #decides one value fro
m get_potential_splits() to split the data set

  split_column_values = data[:, split_column]
  data_below = data[split_column_values <= split_value]
  data_above = data[split_column_values >  split_value]

  return data_below, data_above
```

```
def calculate_entropy(data): #function to calculate entropy for purity
in numbers

  label_column = data[:, -1]
  _, counts = np.unique(label_column, return_counts=True)
  probabilities = counts / counts.sum()
  entropy = sum(probabilities * -np.log2(probabilities))
  return entropy

def calculate_overall_entropy(data_below, data_above): #Function using
above one to calculate total entropy from our two splits

  n = len(data_below) + len(data_above)
  p_data_below = len(data_below) / n
  p_data_above = len(data_above) / n
  overall_entropy = (p_data_below * calculate_entropy(data_below) + p_d
ata_above * calculate_entropy(data_above))
  return overall_entropy

def determine_best_split(data, potential_splits): #after running all sp
lit potentials this function returns the best one wrt entropy

  overall_entropy = 999
  for column_index in potential_splits:
    for value in potential_splits[column_index]:
      data_below, data_above = split_data(data, split_column=column_ind
ex, split_value=value)
      current_overall_entropy = calculate_overall_entropy(data_below, d
ata_above)
      if current_overall_entropy <= overall_entropy:
        overall_entropy = current_overall_entropy
        best_split_column = column_index#return column for classificati
on

        best_split_value = value #return value to split the data

  return best_split_column, best_split_value

# Main Algorithm
def decision_tree_algorithm(df, counter=0, min_samples=2, max_depth=5):
 #grows trees xD i m proud!

  # data preparations
  if counter == 0: #Converting padas (CSV) into numpy [Change file fomr
at]
    global COLUMN_HEADERS
    COLUMN_HEADERS = df.columns
```

```
    data = df.values #seperated column headers and entries into COLUMN_
Headers and data
  else:
    data = df  #saving already converted file

  # base cases
  if (check_purity(data)) or (len(data) < min_samples) or (counter == m
ax_depth):
    classification = classify_data(data)
    return classification #answer

  # recursive part
  else:
    counter += 1

    #calling functions
    potential_splits = get_potential_splits(data)
    split_column, split_value = determine_best_split(data, potential_sp
lits)
    data_below, data_above = split_data(data, split_column, split_value)

    # instantiate sub-tree
    feature_name = COLUMN_HEADERS[split_column]
    question = "{} <= {}".format(feature_name, split_value) #feature_na
me <= split_value || text represntation of determine_best_split()
    sub_tree = {question: []} #this is a tree for dictionary x  array

    # find answers (recursion)
    yes_answer = decision_tree_algorithm(data_below, counter, min_sampl
es, max_depth) #
    no_answer = decision_tree_algorithm(data_above, counter, min_sample
s, max_depth)

    if yes_answer == no_answer:
      sub_tree = yes_answer
    else:
      sub_tree[question].append(yes_answer)
      sub_tree[question].append(no_answer)

    #print(sub_tree)
    return sub_tree

tree = decision_tree_algorithm(train_df, max_depth=3) #final answer
#print(tree)


def classify_example(example, tree): #compare example with our formed t
ree
```

```
  question = list(tree.keys())[0]
  feature_name, comparison_operator, value = question.split() #seperating question string into 3 different variable strings

  # check for answer
  if example[feature_name] <= float(value):
    answer = tree[question][0]
  else:
    answer = tree[question][1]

  # base case
  if not isinstance(answer, dict):
    return answer

  # recursive part
  else:
    residual_tree = answer
    return classify_example(example, residual_tree)

# example = test_df.iloc[8]
# print(example)

# example_ans = classify_example(example, tree)
# print("classification : ", example_ans)

def calculate_accuracy(df, tree): #how

  df["classification"] = df.apply(classify_example, axis=1, args=(tree,))
  df["classification_correct"] = df["classification"] == df["species"]

  accuracy = df["classification_correct"].mean()

  print(df[["species", "classification", "classification_correct"]])

  return accuracy

accuracy = calculate_accuracy(test_df, tree)
print("\naccuracy : ", accuracy)
```

Output:

	species	classification	classification_correct
31	setosa	setosa	True
142	virginica	virginica	True
65	versicolor	versicolor	True
45	setosa	setosa	True
120	virginica	virginica	True
74	versicolor	versicolor	True
53	versicolor	versicolor	True
85	versicolor	versicolor	True
63	versicolor	versicolor	True
108	virginica	virginica	True
122	virginica	virginica	True
75	versicolor	versicolor	True
143	virginica	virginica	True
0	setosa	setosa	True
104	virginica	virginica	True
10	setosa	setosa	True
93	versicolor	versicolor	True
111	virginica	virginica	True
105	virginica	virginica	True
38	setosa	setosa	True
73	versicolor	versicolor	True
20	setosa	setosa	True
23	setosa	setosa	True
15	setosa	setosa	True
77	versicolor	virginica	False
64	versicolor	versicolor	True
112	virginica	virginica	True
71	versicolor	versicolor	True
81	versicolor	versicolor	True
129	virginica	virginica	True

```
accuracy :  0.9666666666666667
```