|  |  |
| --- | --- |
|  | **EXP.NO:5**  **DATE:09-08-2023** |
| **Aim:** Implement Decision-Tree Learning algorithm using python  **Training Dataset:**    **Source Code:**  **data\_loader.py** :  import csv  def read\_data(csv\_filename):  with open(csv\_filename, 'r') as csvfile:  datareader = csv.reader(csvfile, delimiter=',')  headers = next(datareader)  metadata = []  traindata = []  for name in headers:  metadata.append(name)  for row in datareader:  traindata.append(row)  return (metadata, traindata)    # Corrected usage  metadata, traindata = read\_data("tennis.csv")  **decision\_tree.py** :  import numpy as np  import math  from data\_loader import read\_data  class Node:  def \_\_init\_\_(self, attribute):  self.attribute = attribute  self.children = []  self.answer = ""  def \_\_str\_\_(self):  return self.attribute  def subtables(data, col, delete):  dict = {}  items = np.unique(data[:, col])  count = np.zeros((items.shape[0], 1), dtype=np.int32)  for x in range(items.shape[0]):  for y in range(data.shape[0]):  if data[y, col] == items[x]:  count[x] += 1  for x in range(items.shape[0]):  dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")  pos = 0  for y in range(data.shape[0]):  if data[y, col] == items[x]:  dict[items[x]][pos] = data[y]  pos += 1  if delete:  dict[items[x]] = np.delete(dict[items[x]], col, 1)  return items, dict  def entropy(S):  items = np.unique(S)  if items.size == 1:  return 0  counts = np.zeros((items.shape[0], 1))  sums = 0  for x in range(items.shape[0]):  counts[x] = sum(S == items[x]) / (S.size \* 1.0)  for count in counts:  sums += -1 \* count \* math.log(count, 2)  return sums  def gain\_ratio(data, col):  items, dict = subtables(data, col, delete=False)  total\_size = data.shape[0]  entropies = np.zeros((items.shape[0], 1))  intrinsic = np.zeros((items.shape[0], 1))  for x in range(items.shape[0]):  ratio = dict[items[x]].shape[0] / (total\_size \* 1.0)  entropies[x] = ratio \* entropy(dict[items[x]][:, -1])  intrinsic[x] = ratio \* math.log(ratio, 2)  total\_entropy = entropy(data[:, -1])  iv = -1 \* sum(intrinsic)  for x in range(entropies.shape[0]):  total\_entropy -= entropies[x]  return total\_entropy / iv  def create\_node(data, metadata):  if (np.unique(data[:, -1])).size == 1:  node = Node("")  node.answer = np.unique(data[:, -1])[0]  return node  gains = np.zeros((data.shape[1] - 1, 1))  for col in range(data.shape[1] - 1):  gains[col] = gain\_ratio(data, col)  split = np.argmax(gains)  node = Node(metadata[split])  metadata = np.delete(metadata, split, 0)  items, dict = subtables(data, split, delete=True)  for x in range(items.shape[0]):  child = create\_node(dict[items[x]], metadata)  node.children.append((items[x], child))  return node  def empty(size):  s = ""  for x in range(size):  s += " "  return s  def print\_tree(node, level):  if node.answer != "":  print(empty(level), node.answer)  return  print(empty(level), node.attribute)  for value, n in node.children:  print(empty(level + 1), value)  print\_tree(n, level + 2)  metadata, traindata = read\_data("Tennis.csv")  data = np.array(traindata)  node = create\_node(data, metadata)  print\_tree(node, 0)  **Output:**  **outlook**  **overcast**  **b'yes'**  **rain**  **wind**  **b'strong'**  **b'no'**  **b'weak'**  **b'yes'**  **sunny**  **humidity**  **b'high'**  **b'no'**  **b'normal'**  **b'yes** | |