PROGRAM\_3 Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

|  |
| --- |
| **import** pandas **as** pd  **from** pandas **import** DataFrame  df\_tennis = DataFrame.from\_csv('C:\\Users\\ISE\\Desktop\\Python-Decision-Tree-Using-ID3-master\\PlayTennis.csv')  **print**("\n Given Play Tennis Data Set:\n\n", df\_tennis)  df\_tennis.keys()[0] |

Entropy of the Training Data Set

|  |
| --- |
| **def** entropy(probs)**:**  **import math**  **return sum**( [-prob\*math.log(prob, 2) for prob in probs] )  **def** **entropy\_of\_list**(a\_list)**:**    from collections import Counter  cnt = Counter(x for x in a\_list)  num\_instances = len(a\_list)\*1.0  print("\n Number of Instances of the Current Sub Class is{0}:".format(num\_instances ))  probs = [x / num\_instances for x in cnt.values()]  print("\n Classes:",min(cnt),max(cnt))  print(" \n Probabilities of Class {0} is {1}:".format(min(cnt),min(probs)))  print(" \n Probabilities of Class {0} is {1}:".format(max(cnt),max(probs)))  return entropy(probs) # Call Entropy :    **print**("\n INPUT DATA SET FOR ENTROPY CALCULATION:\n", df\_tennis['PlayTennis'])  total\_entropy = entropy\_of\_list(df\_tennis['PlayTennis'])  **print**("\n Total Entropy of PlayTennis Data Set:",total\_entropy) |

Information Gain of Attributes

|  |
| --- |
| **def information\_gain**(df, split\_attribute\_name, target\_attribute\_name, trace=0):  print("Information Gain Calculation of ",split\_attribute\_name)  **df\_split = df.groupby**(split\_attribute\_name)  nobs = **len**(df.index) \* 1.0  **df\_agg\_ent** = **df\_split.agg**({target\_attribute\_name : [entropy\_of\_list, lambda x: len(x)/nobs] })[target\_attribute\_name]  df\_agg\_ent.columns = ['Entropy', 'PropObservations']  new\_entropy = sum( df\_agg\_ent['Entropy'] \* df\_agg\_ent['PropObservations'] )  old\_entropy = entropy\_of\_list(df[target\_attribute\_name])  return old\_entropy - new\_entropy  **print**('Info-gainfor Outlook is :'+str( information\_gain(df\_tennis, 'Outlook', 'PlayTennis')),"\n")  print('\n Info-gain for Humidity is: ' + str( information\_gain(df\_tennis, 'Humidity', 'PlayTennis')),"\n")  print('\n Info-gain for Wind is:' + str( information\_gain(df\_tennis, 'Wind', 'PlayTennis')),"\n")  print('\n Info-gain for Temperature is:' + str( information\_gain(df\_tennis, 'Temperature','PlayTennis')),"\n") |

**ID3 Algorithm**

|  |
| --- |
| **def id3**(df, target\_attribute\_name, attribute\_names, default\_class=None**):**    from collections import Counter  cnt = Counter(x for x in df[target\_attribute\_name])    if len(cnt) == 1:  return next(iter(cnt))    elif **df.empty** or (not attribute\_names**):**  return default\_class    else:    default\_class = max(cnt.keys())    gainz = [information\_gain(df, attr, target\_attribute\_name) for attr in attribute\_names] #  index\_of\_max = gainz.index(max(gainz)) # Index of Best Attribute    best\_attr = attribute\_names[index\_of\_max]      tree = {best\_attr:{}}  remaining\_attribute\_names = [i for i in attribute\_names if i != best\_attr]    for attr\_val, data\_subset in df.groupby(best\_attr):  subtree = id3(data\_subset,  target\_attribute\_name,  remaining\_attribute\_names,  default\_class)  tree[best\_attr][attr\_val] = subtree  return tree |

# Predicting Attributes

|  |
| --- |
| attribute\_names = list(df\_tennis.columns)  print("List of Attributes:", attribute\_names)  attribute\_names.remove('PlayTennis')  print("Predicting Attributes:", attribute\_names) |

# Tree Construction

|  |
| --- |
| from pprint import pprint  tree = id3(df\_tennis,'PlayTennis',attribute\_names)  print("\n\nThe Resultant Decision Tree is :\n")  pprint(tree)  attribute = next(iter(tree))  print("Best Attribute :\n",attribute)  print("Tree Keys:\n",tree[attribute].keys()) |

# Classification Accuracy

|  |
| --- |
| def classify(instance, tree, default=None): # Instance of Play Tennis with Predicted      attribute = next(iter(tree))  print("Key:",tree.keys())  print("Attribute:",attribute)      if instance[attribute] in tree[attribute].keys():  result = tree[attribute][instance[attribute]]  print("Instance Attribute:",instance[attribute],"TreeKeys:",tree[attribute].keys())  if isinstance(result, dict):  return classify(instance, result)  else:  return result  else:  return default |

|  |
| --- |
| df\_tennis['predicted'] = df\_tennis.apply(classify, axis=1, args=(tree,'No') )  # classify func allows for a default arg: when tree doesn't have answer for a particular  # combitation of attribute-values, we can use 'no' as the default guess  print(df\_tennis['predicted'])  print('\n Accuracy is:\n' + str( sum(df\_tennis['PlayTennis']==df\_tennis['predicted'] ) / (1.0\*len(df\_tennis.index)) ))  df\_tennis[['PlayTennis', 'predicted']] |

# Classification Accuracy: Training/Testing Set

|  |
| --- |
| training\_data = df\_tennis.iloc[1:-4] # all but last four instances  test\_data = df\_tennis.iloc[-4:] # just the last four  train\_tree = id3(training\_data, 'PlayTennis', attribute\_names)  test\_data['predicted2'] = test\_data.apply(  classify,  axis=1,  args=(train\_tree,'Yes') )  print ('\n\n Accuracy is : ' + str( sum(test\_data['PlayTennis']==test\_data['predicted2'] ) / (1.0\*len(test\_data.index)) )) |