Source Code:

```
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):
 #TODO: Co jeśli information gain jest zerowe?
 if (np.unique(data[:, -1])).shape[0] == 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.data")
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'
```

OR

```
import pandas as pd
import numpy as np
dataset=
pd.read_csv('playtennis.csv',names=['outlook','temperature','humidity','wind','cla
ss',])
def entropy(target_col):
 elements,counts = np.unique(target_col,return_counts = True)
 entropy = np.sum([(-
counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts)) for i in
range(len(elements))])
 return entropy
def InfoGain(data,split_attribute_name,target_name="class"):
 total_entropy = entropy(data[target_name])
 vals,counts= np.unique(data[split_attribute_name],return_counts=True)
 Weighted_Entropy =
np.sum([(counts[i]/np.sum(counts))*entropy(data.where(data[split_attribute_na
me]==vals[i]).dr
opna()[target_name]) for i in range(len(vals))])
 Information_Gain = total_entropy - Weighted_Entropy
 return Information_Gain

def
ID3(data,originaldata,features,target_attribute_name="class",parent_node_class
= None):
 if len(np.unique(data[target_attribute_name])) <= 1:
 return np.unique(data[target_attribute_name])[0]
 elif len(data)==0:
 return
np.unique(originaldata[target_attribute_name])[np.argmax(np.unique(originalda
ta[target_attribut
e_name],return_counts=True)[1])]
 elif len(features) ==0:
 return parent_node_class
 else:
 parent_node_class =
np.unique(data[target_attribute_name])[np.argmax(np.unique(data[target_attrib
ute_name],return
_counts=True)[1])]
 item_values = [InfoGain(data,feature,target_attribute_name) for feature in
features] #Return
the information gain values for the features in the dataset
 best_feature_index = np.argmax(item_values)
 best_feature = features[best_feature_index]
 tree = {best_feature:{}}
 features = [i for i in features if i != best_feature]
 for value in np.unique(data[best_feature]):
 value = value
 sub_data = data.where(data[best_feature] == value).dropna()
 subtree =
ID3(sub_data,dataset,features,target_attribute_name,parent_node_class)
 tree[best_feature][value] = subtree
 return(tree)
tree = ID3(dataset,dataset,dataset.columns[:-1])
print(' \nDisplay Tree\n',tree)
```

OUTPUT:

```
Display Tree
{'outlook': {'Overcast': 'Yes', 'Rain': {'wind': {'Strong': 'No', 'Weak': 'Yes'}},
'Sunny':
{'humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```