3 Decision tree based ID3, build and classify for appropriate dataset

Import Play Tennis Data

```
from math import log2
from collections import Counter

import pandas as pd
dataframe = pd.read_csv('PlayTennis.csv')
```

```
#piping out the dataframe
dataframe
```

	PlayTennis	Outlook	Temperature	Humidity	Wind
0	No	Sunny	Hot	High	Weak
1	No	Sunny	Hot	High	Strong
2	Yes	Overcast	Hot	High	Weak
3	Yes	Rain	Mild	High	Weak
4	Yes	Rain	Cool	Normal	Weak
5	No	Rain	Cool	Normal	Strong
6	Yes	Overcast	Cool	Normal	Strong
7	No	Sunny	Mild	High	Weak
8	Yes	Sunny	Cool	Normal	Weak
9	Yes	Rain	Mild	Normal	Weak
10	Yes	Sunny	Mild	Normal	Strong
11	Yes	Overcast	Mild	High	Strong
12	Yes	Overcast	Hot	Normal	Weak
13	No	Rain	Mild	High	Strong

Entropy of the Training Data Set

```
def entropy(subframe):

    counter = Counter(subframe)
    num_instances = len(subframe)

    entropy = 0
    for value in counter.values():
        probability = value/num_instances
        entropy = entropy   +  ( -probability * log2(probability) )   # Entropy,  -p*log2*p

    return entropy
```

Information Gain of Attributes

```
total_samples = len(dataframe.index)

# helper function in finding aggregates

def probability(samples):
        return len(samples)/total_samples
```

```
def information_gain(dataframe, split_attribute, target_attribute):

    # Split and Aggregate Data by possible vals of Attribute:

    frame = dataframe.groupby(split_attribute)
    aggregate = frame.agg(  [entropy, probability] )

    aggregate = aggregate[target_attribute]      # since interest is only on target


    new_entropy = sum( aggregate['entropy'] * aggregate['probability'] )
    old_entropy = entropy(dataframe[target_attribute])
    return old_entropy - new_entropy
```

ID3 Algorithm

```
def id3(dataframe, target_attribute, attributes_list, default_class=None):




    counter = Counter(dataframe[target_attribute])              # Atrribute class of YES/NO (binary)


    ''' First check: Is this split of the dataset homogeneous ?   '''

    if len(counter) == 1:
        return next(iter(counter))
                                                                # counter is a dictionary, iterate the dictionary, and release the
first key


        ''' Second check: Is this split of the dataset empty ?  '''

    elif dataframe.empty or (not attributes_list):
        return default_class                                    # Return None for Empty Data Set


        ''' Otherwise: '''
    else:

        # Get Default Value for next recursive call of this function :

        default_class = max(counter)

 #-------------------------------------------------------------------------------------

        # Compute the Information Gain of all the attributes:

        gains = []
        for attribute in attributes_list:
            gains.append(
                            [ information_gain(dataframe, attribute, target_attribute), attribute ]
                        )

                                                                 #appending attribute, will be easy in extracting attribute with ma
xgain


#-------------------------------------------------------------------------------------

        # Choose Best Attribute to split on:
        best_attribute = max(gains)[1]                          # 1 because it's attribute's index

        # Create an empty tree, with best attribute as a node
        tree = { best_attribute : {} }

        attributes_list.remove(best_attribute)                  # since interest is on remaining attributes

 #-------------------------------------------------------------------------------------

        # Split dataset
        #      On each split, recursively call this algorithm.
        # populate the empty tree with subtrees, which are the result of the recursive call

        for attribute_value, data_subset in dataframe.groupby(best_attribute):
            subtree = id3(data_subset, target_attribute, attributes_list, default_class)

            tree[best_attribute][attribute_value] = subtree


        return tree
```

Predicting Attributes

```python
# Get the column headers in dateframe

attributes_list = list(dataframe.columns)
attributes_list.remove('PlayTennis')                          #     since it is not an attribute

print(attribute_list)
```

```
['Outlook', 'Temperature', 'Humidity', 'Wind']
```

Tree Construction

```python
# Run Algorithm:


tree = id3(dataframe, 'PlayTennis', attributes_list.copy() )    #.copy since attributes are removed in recursions


# pretty print

from pprint import  pprint
pprint(tree)
```

```
{'Outlook': {'Overcast': 'Yes',
             'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

Classification Accuracy

```python
#classifying function

def classify(instance, tree, default=None):


    attribute = next(iter(tree))                     # Outlook/Humidity/Wind

    subtree =  tree[attribute]

    if instance[attribute] in subtree:               # Value of the attributs in  set of Tree keys
        result = subtree[ instance[attribute] ]

        if isinstance(result, dict):                 # if it is a tree(dictionary), go deeper
            return classify(instance, result)
        else:
            return result                            # this is a label
    else:
        return default
```

```python
# creating a new column "predicted" in dataframe by applying "classify" along rows

dataframe['predicted'] = dataframe.apply(classify, axis=1, args=(tree,'No') )


# Accuracy, no_of_correct_predictions / total_samples

accuracy =  sum( dataframe['PlayTennis'] == dataframe['predicted'] ) / len(dataframe.index)
print('Accuracy is: ', accuracy )


dataframe
```

```
Accuracy is:  1.0
```

	PlayTennis	Outlook	Temperature	Humidity	Wind	predicted
0	No	Sunny	Hot	High	Weak	No
1	No	Sunny	Hot	High	Strong	No
2	Yes	Overcast	Hot	High	Weak	Yes
3	Yes	Rain	Mild	High	Weak	Yes
4	Yes	Rain	Cool	Normal	Weak	Yes
5	No	Rain	Cool	Normal	Strong	No
6	Yes	Overcast	Cool	Normal	Strong	Yes
7	No	Sunny	Mild	High	Weak	No
8	Yes	Sunny	Cool	Normal	Weak	Yes
9	Yes	Rain	Mild	Normal	Weak	Yes
10	Yes	Sunny	Mild	Normal	Strong	Yes
11	Yes	Overcast	Mild	High	Strong	Yes
12	Yes	Overcast	Hot	Normal	Weak	Yes
13	No	Rain	Mild	High	Strong	No

Classification Accuracy: Training/Testing Set

```python
                                                          #.copy() to ignore a warning
training_data = dataframe.iloc[:-4].copy()                # all but last four instances and column headers
test_data  = dataframe.iloc[-4:].copy()                   # just the last four



train_tree = id3(training_data, 'PlayTennis', attributes_list.copy() )    #Note : .copy

test_data['predicted2'] = test_data.apply(                               # <---- test_data source
                                          classify,
                                          axis=1,
                                          args=(train_tree, 'Yes') )       # <---- train_data tree

accuracy =  sum( test_data['PlayTennis'] == test_data['predicted2'] ) / len(test_data.index)
print('Accuracy is over test data : ', accuracy )
```

```
Accuracy is over test data :  0.75
```

```python
#P.S : Dataset
for line in open('PlayTennis.csv').readlines():
    print("{0:>10},{1:>10},{2:>10},{3:>10}".format(*line.strip().split(',')))
```

```
PlayTennis,   Outlook,Temperature,  Humidity
        No,     Sunny,       Hot,      High
        No,     Sunny,       Hot,      High
       Yes,  Overcast,       Hot,      High
       Yes,      Rain,      Mild,      High
       Yes,      Rain,      Cool,    Normal
        No,      Rain,      Cool,    Normal
       Yes,  Overcast,      Cool,    Normal
        No,     Sunny,      Mild,      High
       Yes,     Sunny,      Cool,    Normal
       Yes,      Rain,      Mild,    Normal
       Yes,     Sunny,      Mild,    Normal
       Yes,  Overcast,      Mild,      High
       Yes,  Overcast,       Hot,    Normal
        No,      Rain,      Mild,      High
```