ID3_weather

October 15, 2020

[1]:
```python
import pandas as pd
```

[2]:
```python
import pandas as pd
import numpy as np
from pprint import pprint
```

[3]:
```python
dataset = pd.read_csv('data3.csv')
```

[4]:
```python
def entropy(target_col):
    """
    Calculate the entropy of a dataset.
    The only parameter of this function is the target_col parameter which␣
 ↪specifies the target column
    """
    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
```

[5]:
```python
def InfoGain(data,split_attribute_name,target_name="Target"):
    """
    Calculate the information gain of a dataset. This function takes three␣
 ↪parameters:
    1. data = The dataset for whose feature the IG should be calculated
    2. split_attribute_name = the name of the feature for which the information␣
 ↪gain should be calculated
    3. target_name = the name of the target feature. The default for this␣
 ↪example is "class"
    """
    #Calculate the entropy of the total dataset
    total_entropy = entropy(data[target_name])

    ##Calculate the entropy of the dataset

    #Calculate the values and the corresponding counts for the split attribute
    vals,counts= np.unique(data[split_attribute_name],return_counts=True)
```

```
    #Calculate the weighted entropy
    Weighted_Entropy = np.sum([(counts[i]/np.sum(counts))*entropy(data.
↪where(data[split_attribute_name]==vals[i]).dropna()[target_name]) for i in␣
↪range(len(vals))])

    #Calculate the information gain
    Information_Gain = total_entropy - Weighted_Entropy
    return Information_Gain
```

[6]:
```
def␣
↪ID3(data,originaldata,features,target_attribute_name="Target",parent_node_class␣
↪= None):
    """
    ID3 Algorithm: This function takes five paramters:
    1. data = the data for which the ID3 algorithm should be run --> In the␣
↪first run this equals the total dataset

    2. originaldata = This is the original dataset needed to calculate the mode␣
↪target feature value of the original dataset
    in the case the dataset delivered by the first parameter is empty

    3. features = the feature space of the dataset . This is needed for the␣
↪recursive call since during the tree growing process
    we have to remove features from our dataset --> Splitting at each node

    4. target_attribute_name = the name of the target attribute

    5. parent_node_class = This is the value or class of the mode target␣
↪feature value of the parent node for a specific node. This is
    also needed for the recursive call since if the splitting leads to a␣
↪situation that there are no more features left in the feature
    space, we want to return the mode target feature value of the direct parent␣
↪node.
    """
    #Define the stopping criteria --> If one of this is satisfied, we want to␣
↪return a leaf node#

    #If all target_values have the same value, return this value
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]

    #If the dataset is empty, return the mode target feature value in the␣
↪original dataset
    elif len(data)==0:
        return np.unique(originaldata[target_attribute_name])[np.argmax(np.
↪unique(originaldata[target_attribute_name],return_counts=True)[1])]
```

```
    #If the feature space is empty, return the mode target feature value of the␣
↪direct parent node --> Note that
    #the direct parent node is that node which has called the current run of␣
↪the ID3 algorithm and hence
    #the mode target feature value is stored in the parent_node_class variable.

    elif len(features) ==0:
        return parent_node_class

    #If none of the above holds true, grow the tree!

    else:
        #Set the default value for this node --> The mode target feature value␣
↪of the current node
        parent_node_class = np.unique(data[target_attribute_name])[
            np.argmax(np.
↪unique(data[target_attribute_name],return_counts=True)[1])]

        #Select the feature which best splits the dataset
        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]

        #Create the tree structure. The root gets the name of the feature␣
↪(best_feature) with the maximum information
        #gain in the first run
        tree = {best_feature:{}}


        #Remove the feature with the best inforamtion gain from the feature␣
↪space
        features = [i for i in features if i != best_feature]

        #Grow a branch under the root node for each possible value of the root␣
↪node feature

        for value in np.unique(data[best_feature]):
            value = value
            #Split the dataset along the value of the feature with the largest␣
↪information gain and therwith create sub_datasets
            sub_data = data.where(data[best_feature] == value).dropna()
```

```
            #Call the ID3 algorithm for each of those sub_datasets with the new␣
↪parameters --> Here the recursion comes in!
            subtree =␣
↪ID3(sub_data,dataset,features,target_attribute_name,parent_node_class)

            #Add the sub tree, grown from the sub_dataset to the tree under the␣
↪root node
            tree[best_feature][value] = subtree

        return(tree)
```

[7]:
```
def train_test_split(dataset):
    training_data = dataset.iloc[:80].reset_index(drop=True)#We drop the index␣
↪respectively relabel the index
    #starting form 0, because we do not want to run into errors regarding the␣
↪row labels / indexes
    testing_data = dataset.iloc[80:].reset_index(drop=True)
    return training_data,testing_data

training_data = train_test_split(dataset)[0]
testing_data = train_test_split(dataset)[1]
```

[8]:
```
tree = ID3(training_data,training_data,training_data.columns[:-1])
pprint(tree)
```

```
{'Outlook': {'overcast': 'yes ',
             'rain': {'Wind': {'strong': {'Temperature': {'cool': 'no ',
                                                          'mild': 'no'}},
                               'weak': 'yes '}},
             'sunny': {'Humidity': {'high': 'no ', 'normal': 'yes '}}}}
```

[ ]:
```
```