Experiment 3 - Decision Tree Using ID3

April 30, 2023

1 Experiment Details

1.1 Submitted By

Desh Iyer, 500081889, Year III, AI/ML(H), B5

1.2 Problem Statement

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.

1.3 Theory

The ID3 (Iterative Dichotomiser 3) algorithm is a decision tree based algorithm used for classification in machine learning. It works by building a decision tree incrementally, using a heuristic called information gain to select the best attribute to split the data at each node. The algorithm starts with the entire training set and recursively splits it into smaller subsets based on the selected attributes until all the data in a subset belong to the same class. The decision tree can then be used for classification by traversing it from the root node to the leaf node that corresponds to the predicted class.

Here are the terms used in the description of the algorithm.

- A decision tree is a tree-like model used for decision-making in machine learning.
- A node in the decision tree represents a subset of the data.
- An attribute is a characteristic of the data that can be used to split it into subsets.
- A leaf node represents a class label.
- Information gain is a heuristic used to select the best attribute to split the data.

1.4 Advantages

- The algorithm is simple and easy to understand.
- The resulting decision tree is easy to interpret and can be used to explain the classification results.
- The algorithm can handle both categorical and numerical data.

1.5 Limitations

- The algorithm tends to overfit the training data, which can lead to poor generalization performance.

- The algorithm does not handle missing values well.
- The algorithm is sensitive to noisy data and outliers.

1.6 Pseudocode

In this pseudocode, `data` represents the training data, `attributes` represents the set of attributes in the data, and `target_attribute` represents the class label. The function returns a decision tree that can be used for classification.

```
function ID3(data, attributes, target_attribute)
    if all examples in data are of the same class:
        return a leaf node with the class label
    else if attributes is empty:
        return a leaf node with the majority class label
    else:
        best_attribute ← the attribute with the highest information gain
        tree ← a new decision tree with root node best_attribute
        for each value vi of best_attribute do
        subset ← the subset of examples in data with value vi for best_attribute
        if subset is empty:
            subtree ← a leaf node with the majority class label
        else:
            subtree ← ID3(subset, attributes - {best_attribute}, target_attribute)
        add subtree to tree as a child node with label vi
        return tree
```

2 Import Libraries

In the ID3 algorithm, decision trees are calculated using the concept of entropy and information gain.

[ ]:
```python
import pandas as pd
import numpy as np

# eps for making value a bit greater than 0 later on
eps = np.finfo(float).eps

from numpy import log2 as log
```

3 Creating Dataset

[ ]:
```python
dataset = {'Taste':
 ↪['Salty','Spicy','Spicy','Spicy','Spicy','Sweet','Salty','Sweet','Spicy','Salty'],
       'Temperature':
 ↪['Hot','Hot','Hot','Cold','Hot','Cold','Cold','Hot','Cold','Hot'],
       'Texture':
 ↪['Soft','Soft','Hard','Hard','Hard','Soft','Soft','Soft','Soft','Hard'],
```

```
        'Eat':['No','No','Yes','No','Yes','Yes','No','Yes','Yes','Yes']}
```

[ ]:
```
df = pd.DataFrame(dataset,columns=['Taste','Temperature','Texture','Eat'])
df
```

[ ]:

	Taste	Temperature	Texture	Eat
0	Salty	Hot	Soft	No
1	Spicy	Hot	Soft	No
2	Spicy	Hot	Hard	Yes
3	Spicy	Cold	Hard	No
4	Spicy	Hot	Hard	Yes
5	Sweet	Cold	Soft	Yes
6	Salty	Cold	Soft	No
7	Sweet	Hot	Soft	Yes
8	Spicy	Cold	Soft	Yes
9	Salty	Hot	Hard	Yes

4 Function to Calculate Entropy of a Label

[ ]:
```
def find_entropy(df):
    '''
    Function to calculate the entropy of a label
    '''
    Class = df.keys()[-1]
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value] / len(df[Class])
        entropy += -fraction * np.log2(fraction)
    return entropy
```

5 Function to Calculate Entropy of all Features

[ ]:
```
def find_entropy_attribute(df,attribute):
    '''
    Function to calculate the entropy of all features.
    '''
    Class = df.keys()[-1]
    target_variables = df[Class].unique()
    variables = df[attribute].unique()
    entropy2 = 0
    for variable in variables:
        entropy = 0
        for target_variable in target_variables:
```

```
                num =␣
↪len(df[attribute][df[attribute]==variable][df[Class]==target_variable])
                den = len(df[attribute][df[attribute]==variable])
                fraction = num/(den + eps)
                entropy += -fraction * log(fraction + eps)
        fraction2 = den / len(df)
        entropy2 += -fraction2 * entropy
    return abs(entropy2)
```

6 Function to Find the Feature with the Highest Information Gain

[ ]:
```
def find_winner(df):
    '''
    Function to find the feature with the highest information gain.
    '''
    IG = []
    for key in df.keys()[:-1]:
    # Entropy_att.append(find_entropy_attribute(df,key))
        IG.append(find_entropy(df) - find_entropy_attribute(df,key))
    return df.keys()[:-1][np.argmax(IG)]
```

7 Function to Get a Sub-table of Met Conditions

[ ]:
```
def get_subTable(df, node, value):
    '''
    Function to get a subTable of met conditions.

    node: Column name
    value: Unique value of the column
    '''
    return df[df[node] == value].reset_index(drop=True)
```

8 Function to Build the ID3 Decision Tree

[ ]:
```
def buildTree(df,tree=None):
    '''
    Function to build the ID3 Decision Tree.
    '''
    Class = df.keys()[-1]
    #Here we build our decision tree

    #Get attribute with maximum information gain
    node = find_winner(df)
```

```
    #Get distinct value of that attribute e.g Salary is node and Low,Med and␣
↪High are values
    attValue = np.unique(df[node])

    #Create an empty dictionary to create tree
    if tree is None:
        tree={}
        tree[node] = {}

   #We make loop to construct a tree by calling this function recursively.
    #In this we check if the subset is pure and stops if it is pure.

    for value in attValue:

        subTable = get_subTable(df,node,value)
        clValue,counts = np.unique(subTable['Eat'],return_counts=True)

        if len(counts)==1:#Checking purity of subset
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subTable) #Calling the function␣
↪recursively

    return tree
```

9 Building the Decision Tree

[ ]:
```
tree = buildTree(df)
```

The tree splits are as follows,

[ ]:
```
import pprint
pprint.pprint(tree)
```

```
{'Taste': {'Salty': {'Texture': {'Hard': 'Yes', 'Soft': 'No'}},
           'Spicy': {'Temperature': {'Cold': {'Texture': {'Hard': 'No',
                                                          'Soft': 'Yes'}},
                                     'Hot': {'Texture': {'Hard': 'Yes',
                                                         'Soft': 'No'}}}},
           'Sweet': 'Yes'}}
```

10 Function to Predict for Any Input

Now, for prediction we go through each node of the tree to find the output.

[ ]:
```
def predict(inst,tree):
    '''
    Function to predict for any input variable.
    '''
    # Recursively we go through the tree that we built earlier

    for nodes in tree.keys():

        value = inst[nodes]
        tree = tree[nodes][value]
        prediction = 0

        if type(tree) is dict:
            prediction = predict(inst, tree)
        else:
            prediction = tree
            break;

    return prediction
```

11 Predicting on Test Data

[ ]:
```
data = {'Taste':'Salty','Temperature':'Cold','Texture':'Hard'}
```

[ ]:
```
inst = pd.Series(data)
```

[ ]:
```
prediction = predict(inst,tree)
prediction
```

[ ]:
```
'Yes'
```
