Web Mining – Lab 8

Implementation of Decision Trees

By Abhijeet Ambadekar (16BCE1156)

Problem Statement:
To implement a decision tree classifier on categorical data.

Porgram Code:

```
import numpy as np
import pandas as pd

from pprint import pprint
from math import log

class DecisionTree():
	def __init__(self):
		self.tree = {}

	def calc_entropy(self, target_vals):
		entropy = 0.0
		tot_size = target_vals.shape[0]
		target_value_counts = target_vals.value_counts()
		for val in target_vals.unique():
			size = target_value_counts[val]
			entropy -= log(size/tot_size) * size/tot_size

		return entropy



	def fit(self, data, target):
		self.data = data
		self.target = target
		outcom_counts = self.data.loc[:, self.target].unique().shape[0]

	def generate_subtree_tree(self, data = pd.DataFrame([]), parent=-1, tree = 
{}, parentval = None):
		if data.empty:
			data = self.data

		entropy = {}
		infoGains = {}

		if tree.keys():
			nodenumber = max(tree.keys()) + 1
		else:
```

```
            nodenumber = 0

        for col in data.columns:
            #  If column is target, the computed entropy is the entropy is the current dataset
            if col == self.target:
                entropy[col] = self.calc_entropy(data.loc[:, col])
            # Else each column has different values having different target values, for which entropies are computed indivisually
            # And their weighted sum in computed
            else:
                tot_size = data.shape[0]
                entropy[col] = 0.0
                for val in data.loc[:, col].unique():
                    subset_data = data.loc[(data[col] == val), self.target]
                    size = subset_data.shape[0]
                    entropy[col] += self.calc_entropy(subset_data) * size / tot_size

        for col in data.columns:
            if col == self.target:
                continue
            infoGains[col] = entropy[self.target] - entropy[col]

        # pprint(infoGains)
        # Based on the maximum InfoGain the dataset is split on the basis of that column
        # And the function is performed on indivisual subsets recursively
        maxkey = max(infoGains, key = lambda key: infoGains[key])
        # print(maxkey)

        if infoGains[maxkey] == 0:
            tree[nodenumber] = (parent, data.iloc[0,:][self.target], True, parentval)
            return tree

        else:
            # if parent == -1:
            tree[nodenumber] = (parent, maxkey, False, parentval)
            # else:
                # tree[nodenumber] = (parent, maxkey, False, )

            for curr_val in data.loc[:, maxkey].unique():
                subset_data = data.loc[(data[maxkey] == curr_val), :]
                tree = self.generate_subtree_tree(subset_data, nodenumber, tree, curr_val)

        if parent == -1:
            self.tree = tree
        else:
```

```
                return tree


        def print_tree(self):
            pprint(self.tree)
            print()

        def show_tree(self, pnode = 0, level = 0):
            if self.tree[pnode][3] == None:
                strdata = str(pnode) + " split at " + str(self.tree[pnode][1])
            else:
                if not self.tree[pnode][2]:
                    strdata = str(pnode) + " parent value = " +
str(self.tree[pnode][3]) + " split at " + str(self.tree[pnode][1])
                else:
                    strdata = str(pnode) + " parent value = " +
str(self.tree[pnode][3]) + " results at target: " + str(self.tree[pnode][1])
            print(" "  * 2 * level, '-' * (level), strdata, sep='')
            for node in [k for k in self.tree if self.tree[k][0] == pnode]:
                self.show_tree(node, level+1)


# datamat = [
#     ["val1", "val3", "val5", "no"],
#     ["val1", "val4", "val5", "yes"],
#     ["val2", "val3", "val5", "yes"],
#     ["val2", "val4", "val6", "no"],
# ]
# datacols = ["param1", "param2", "param3", "target"]

# data = pd.DataFrame(datamat)
# data.columns = datacols


data = pd.read_csv("data.csv")
target = data.columns[-1]

dcc = DecisionTree()
dcc.fit(data, target)
dcc.generate_subtree_tree()
dcc.print_tree()
dcc.show_tree()
```

Data.csv used:

```
WM_L9_1156.py        data.csv
Outlook,Temp,Humidity,Wind,PlayTennis
Sunny,Hot,High,Weak,No
Sunny,Hot,High,Strong,No
Overcast,Hot,High,Weak,Yes
Rain,Mild,High,Weak,Yes
Rain,Cool,Normal,Weak,Yes
Rain,Cool,Normal,Strong,No
Overcast,Cool,Normal,Strong,Yes
Sunny,Mild,High,Weak,No
Sunny,Cool,Normal,Weak,Yes
Rain,Mild,Normal,Weak,Yes
Sunny,Mild,Normal,Strong,Yes
Overcast,Mild,High,Strong,Yes
Overcast,Hot,Normal,Weak,Yes
Rain,Mild,High,Strong,No
```

Outlook:

```
/media/anonymous/Work/Vit/Semester 5/WM/Lab/L8_DecisionTrees  echo 16BCE1156
16BCE1156
/media/anonymous/Work/Vit/Semester 5/WM/Lab/L8_DecisionTrees  python3 WM_L9_1156.py
{0: (-1, 'Outlook', False, None),
 1: (0, 'Humidity', False, 'Sunny'),
 2: (1, 'No', True, 'High'),
 3: (1, 'Yes', True, 'Normal'),
 4: (0, 'Yes', True, 'Overcast'),
 5: (0, 'Wind', False, 'Rain'),
 6: (5, 'Yes', True, 'Weak'),
 7: (5, 'No', True, 'Strong')}

0 split at Outlook
 -1 parent value = Sunny split at Humidity
   --2 parent value = High results at target: No
   --3 parent value = Normal results at target: Yes
 -4 parent value = Overcast results at target: Yes
 -5 parent value = Rain split at Wind
   --6 parent value = Weak results at target: Yes
   --7 parent value = Strong results at target: No
/media/anonymous/Work/Vit/Semester 5/WM/Lab/L8_DecisionTrees
```