Course name: Data Science(ITE4005)
Student id: 2016024793
Name: 김유진

Data Science Programming Assignment #2

PROGRAMMING ASSIGNMENT #2 – DECISION TREE

ENVIRONMENT

OS: Windows 10

Language: python 3.6.8

Used module: math, sys, Counter and defaultdict from collections, partial from functools

```
import math
import sys
from collections import Counter, defaultdict
from functools import partial
```

SUMMARY OF ALGORITHM

In order to build tree easily, dictionary type was used to represent a tuple. Then, class attribute was paired with each tuple in training dataset.

I implemented decision tree which uses information gain or gain ratio and both values were tested after implementation.

Using gain ratio split subsets more evenly than using information gain since it is normalized version of information gain. Also, when tested with given data, gain ration turned out to perform better. Lastly, gini index was not appropriate for the given datasets since gini index is used only for binary splits.

For these reasons, gain ratio was chosen as the criteria for choosing best attribute.

DETAILED DESCRIPTIONS

1. **Read and save training dataset and test dataset (load_train_data, load_test_data)**

Description: function that reads data from given file. Each tuple is turned into a dictionary and given class attribute is then paired with the dictionary.

```python
# read train data file
# return list of attributes and following categories.
def load_train_data(filename):
    f = open(filename, "r")
    attr = []
    cat = []
    data = []

    attr = f.readline().strip("\n").split("\t")
    for _ in range(len(attr)):
        cat.append([])
    while True:
        line = f.readline()
        if not line:
            break
        t = line.strip("\n").split("\t")
        tmp = dict()

        for i in range(len(attr)):
            if not t[i] in cat[i]:
                cat[i].append(t[i])
            if i != len(attr)-1:
                tmp[attr[i]] = t[i]
        data.append((tmp, t[-1]))
    f.close()
    return data, attr, cat

def load_test_data(filename):
    f = open(filename, "r")
    attr = []
    data = []

    attr = f.readline().strip("\n").split("\t")
    while True:
        line = f.readline()
        if not line:
            break

        t = line.strip("\n").split("\t")
        tmp = dict()
        for i in range(len(t)):
            tmp[attr[i]] = t[i]
        data.append(tmp)

    f.close()
    return data
```

returned training data would look like this:

```
[({"age":"<=30", "income":"high", "student":"no", "credit_rating":"fair"}, no)
({"age":"<=30", "income":"high", "student":"no", "credit_rating":"excellent"}, no)
({"age":"31...40", "income":"high", "student":"no", "credit_rating":"fair"}, yes)
....
]
```

2. **Functions for calculating information gain and gain ratio**

Following functions are used when splitting input data into subsets using specific attributes. Information gain and gain ratio was both implemented.

```
# return entropy value using list of probabilities
def entropy(probabilities):
    return sum(-p * math.log(p, 2) for p in probabilities.values() if p is not 0)

# return a dictionary of probabilities by counting labels
#  for example, {'no':0.5, 'yes':0.5}
def class_probabilities(labels):
    total_cnt = len(labels)
    return dict([(key, value / total_cnt) for key, value in
Counter(labels).items()])

# return information gain of labeled dataset
def info_gain(labeled_data):
    labels = [label for _, label in labeled_data]
    probabilities = class_probabilities(labels)
    return entropy(probabilities)

# returns information gain of subsets
def partition_info_gain(subsets):
    total_cnt = sum(len(subset) for subset in subsets)
    return sum(info_gain(subset) * len(subset) / total_cnt for subset in subsets)

# returns split information of subsets.
def partition_split_info(subsets):
    total_cnt = sum(len(subset) for subset in subsets)
    return sum(-1 * len(s) / total_cnt * math.log(len(s)/total_cnt, 2) for s in
subsets)

def partition_by(inputs, attribute):
    # divide inputs into subsets by the attribute
    groups = defaultdict(list)
    for i in inputs:
        key = i[0][attribute]
        groups[key].append(i)
    return groups
```

```python
# return information gain when splited by given attribute
def partition_info_gain_by(inputs, attribute):
    partitions = partition_by(inputs, attribute)
    infoD = info_gain(inputs)
    return infoD - partition_info_gain(partitions.values())

# return gain ratio when splited by given attribute
def partition_gain_ratio_by(inputs, attribute):
    partitions = partition_by(inputs, attribute)
    gain = partition_info_gain_by(inputs, attribute)
    return gain / partition_split_info(partitions.values())
```

3. Build Tree

Build tree recursively. The tree is a tuple consisting of best attribute and subtrees. Attribute value and class attribute value is kept as dictionary type when dataset is split by a specific attribute. Stop conditions were 1) if all input data has same data, return the class label. 2) if there was no more attribute candidate, return a class label by majority voting.

Node with a ‘None’ key value was added in order handle exceptions.

(line 132: choose best attribute using information gain)

```python
# build tree with training dataset
def build_tree(inputs, split_candidates=None):
    def majority(num_class):
        maxKey = None
        maxVal = -1
        for key, value in num_class.items():
            if maxVal < value:
                maxKey, maxVal = key, value
        return maxKey

    if split_candidates is None:
        split_candidates = inputs[0][0].keys()

    num_inputs = len(inputs)
    num_class = Counter([label for _,label in inputs])

    # stop conditions
    # 1) if all tuples have same class labels, return that class label
    for key, value in num_class.items():
        if value == num_inputs:
            return key

    # 2) if there is no more attributes to select,
    #    return a class label by majority voting
    if not split_candidates:
        return majority(num_class)
    # choose best attribute that has largest information gain
    # best_attr = max(split_candidates, key=partial(partition_info_gain_by, inputs))

    # choose best_attribute that has largest gain ratio
    best_attr = max(split_candidates, key=partial(partition_gain_ratio_by, inputs))
    partitions = partition_by(inputs, best_attr)
    new_candidates = [a for a in split_candidates if a != best_attr]

    # build tree recursively
    subtrees = {attr_value : build_tree(subset, new_candidates)
                    for attr_value, subset in partitions.items()}
    # for exception handling
    subtrees[None] = majority(num_class)
    return(best_attr, subtrees)
```

4. Classify

Tree that was built in step 3 is then used to classify other data without class attribute value. This function recursively calls itself until the leaf node is reached. A class attribute value is returned in the end.

```
# classify test dataset recursively
def classify(tree, input, labels):
    # if it is a leaf node, return a value
    if tree in labels:
        return tree

    # otherwise, reach to next node
    attr, subtree_dict = tree
    subtree_key = input.get(attr)

    if subtree_key not in subtree_dict:
        subtree_key = None

    subtree = subtree_dict[subtree_key]

    return classify(subtree, input, labels)
```

5. Print result

Finally, test dataset is classified using classify function and built tree, and finally printed on output file as the given format.

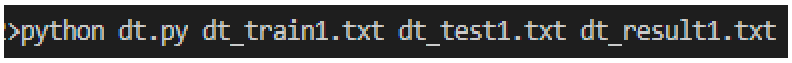
```
def print_result(tree, testData, labels, filename):
    f = open(filename, 'w')

    for a in attr:
        f.write("{}\t".format(a))
    f.write('\n')

    for t in testData:
        for value in t.values():
            f.write("{}\t".format(value))
        result = classify(tree, t, labels)
        f.write("{}\n".format(result))
    f.close()

if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("need 3 arguments")
    else:
        trainfile = sys.argv[1]
        testfile = sys.argv[2]
        outputfile = sys.argv[3]

        data, attr, cat = load_train_data(trainfile)
        tree = build_tree(data)
        testData = load_test_data(testfile)
        print_result(tree, testData, cat[-1], outputfile)
```

COMPILATION

Please use the command below.

python dt.py [*train dataset file name*] [*test dataset file name*] [*output file name*]

For example,



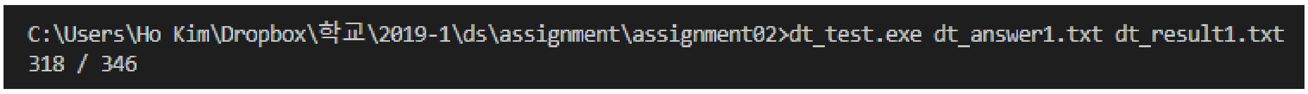
```
>python dt.py dt_train1.txt dt_test1.txt dt_result1.txt
```

Test results

As I explained above, splitting data with gain ratio performed slightly better. Here are the results of tests.

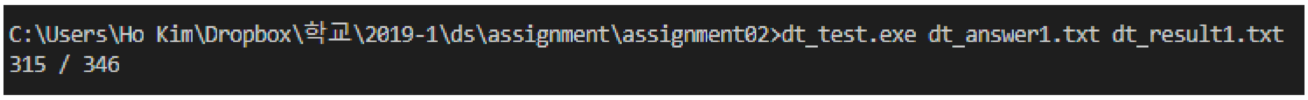
318 out of 346 datasets was classified correctly when attribute with maximum gain ratio was chosen. 315 out of 346 was correctly classified using information gain.

<when gain ratio was used>



```
C:\Users\Ho Kim\Dropbox\학교\2019-1\ds\assignment\assignment02>dt_test.exe dt_answer1.txt dt_result1.txt
318 / 346
```

<when information gain was used>



```
C:\Users\Ho Kim\Dropbox\학교\2019-1\ds\assignment\assignment02>dt_test.exe dt_answer1.txt dt_result1.txt
315 / 346
```