Data Science Assignment #2

# Build a decision tree

2015004857 이영수

## 1) Summary of algorithm

- Tree is constructed in a top-down recursive divide-and-conquer manner
- Tree is automatically built by training
- Each branch of tree means attribute and domain of it
- Measure of attribute selection is "Information Gain (ID3/C4.5)"
- There are 3 cases of deciding result
  - No samples left -> Majority voting by parent nodes' dataset, or choosing first one in result domains
  - No remaining attributes -> Majority voting
  - All samples for given node belong to the same class -> Returns the corresponding value

## 2) Code Description

Codes are written by Python3.


Project consists of python file that implements decision tree, test files, and dataset for testing.

**dt.py**

There is one global function, named `head_and_body`, that gets file as param and return tuple of head information and datasets.

```
"""
@author Prev (prevdev@gmail.com)
"""
import sys
import math


def head_and_body(file):
    """ Read file and convert to (head, body) data

    :param file: Opened file stream
    :return: Tuple of (head, body)
    """
    head = None
    body = []

    for line in file.readlines():
        if line[-1] == '\n': line = line[0:-1]
        t = line.split('\t')

        if head is None:
            # First line
            head = t
        else:
            body.append(t)

    return head, body
```

And there is one class, DescisionTree, that implements building tree and test files. `init` function of this class gets training_set to build tree. Feature for testing files is implementated in `test_files` function, not in `__init__` function.

```
class DecisionTree :

    DOMAINS = {
        'age':    ('<=30', '31...40', '>40'),
        'income': ('high', 'medium', 'low'),
        'student': ('yes', 'no'),
        'credit_rating': ('fair', 'excellent'),
        'Class:buys_computer': ('yes', 'no'),

        'buying': ('vhigh', 'high', 'med', 'low'),
        'maint': ('vhigh', 'high', 'med', 'low'),
        'doors': ('2', '3', '4', '5more'),
        'persons': ('2', '4', 'more'),
        'lug_boot': ('small', 'med', 'big'),
        'safety': ('low', 'med', 'high'),
        'car_evaluation': ('unacc', 'acc', 'good', 'vgood'),
    }

    def __init__(self, training_set) :
        """ Initialize apriori

        :param training_set: Opened file object to train
        """

        self.head, tuples = head_and_body(training_set)
        self.tree = self.maketree(self.head, tuples)

```

`testfile` function gets test_set and output_file. It reads tuples from test_set, run test (predict result), and write predicted results to output_file.

```
59      def testfile(self, test_set, output_file):
60          """ Test dataset in files
61
62          :param test_set: Opened file object to tests
63          :param output_file: Opened file object to write
64          """
65          n_head, tuples = head_and_body(test_set)
66
67          output_file.write('\t'.join(self.head) + '\n')
68
69          for tuple in tuples:
70              d = {}
71              for index, attr in enumerate(n_head):
72                  d[attr] = tuple[index]
73
74              rst = self.test(d)
75              output_file.write('%s\t%s\n' % ('\t'.join(tuple), rst))
```

`test` function gets dictionary-type data (just one tuple, not the set of tuples) and returns predicted value by tree built in `__init__` function. It goes down the tree until the leaf comes out. Leaf is represented as numeric type, and non-leaf is represented as tuple type.

```
78      def test(self, data):
79          """ Test data with trained data (tree)
80
81          :param data: Dictionary data like {attr1: val1, attr2: val2, ...}
82          :return: Predicted value of result attr
83          """
84          cur = self.tree
85
86          while type(cur) == tuple:
87              # `cur` will be tuple if tree is remain (non-leaf)
88              attr, rst = cur
89              cur = rst[data[attr]]
90
91          return cur
```

`maketree` is part for building tree. It gets head, tuples, and prev_tuples param that is generated from recursive call. As I said earlier, leaf is represented as numeric type, and non-leaf is represented as tuple type like comment of below code.

```
94      def maketree(self, head, tuples, prev_tuples=None):
95          """ Make decision tree
96
97          :param head: Attribute(class) set
98          :param tuples: Data set
99          :param prev_tuples: Data set used in previous step (by recursive call)
100         :return: Tree used to decide something
101                 ex) ('age', {
102                         '31...40': 'yes',
103                         '<=30': ('student', {'no': 'no', 'yes': 'yes'}),
104                         '>40': ('credit_rating', {'excellent': 'no', 'fair': 'yes'})
105                     })
106         """
```

There are 3 cases of deciding result. First is that there are no samples left, and next is that there are no remaining attributes on this recursion, and third is that all samples for given node belong to the same class. In first case, it uses majority voting if there are some tuples on previous step, and if not, chooses first one in result domains. In second case, it uses majority voting, and last case returns the value samples belong to.

```
107         if len(tuples) == 0:
108             # There are no samples left
109             if prev_tuples:
110                 # If there are tuples on previous step, vote by majority of it
111                 result_vals = [row[-1] for row in prev_tuples]
112                 return max(set(result_vals), key=result_vals.count)  # majority voting
113             else:
114                 return self.DOMAINS[head[-1]][0]
115 
116         result_vals = [row[-1] for row in tuples]
117 
118         if len(head) == 1:
119             # There are no remaining attributes
120             return max(set(result_vals), key=result_vals.count) # majority voting
121 
122         if result_vals.count(result_vals[0]) == len(result_vals):
123             # All samples for a given node belong to the same class
124             return result_vals[0]
```

If these three cases are not applied, get highest gain attribute on this step (Information Gain <ID3/C4.5>), and divide branches with this attribute. In this case, data structure of tree is looks like below.

```
arrtibute: {
   domain1: <something>,
   domain2: <something>,
   ...
   domainN: <something>
}
```

```
127         gained_attr, data = self._highest_gain_attr(head, tuples)
128 
129         idx = head.index(gained_attr)
130         n_head = head[0:idx] + head[idx + 1:]
131 
132         ret = {}
133         for domain, _tuples in data.items():
134             n_tuples = [t[0:idx] + t[idx + 1:] for t in _tuples]
135             ret[domain] = self.maketree(n_head, n_tuples, tuples)
136 
137         return (gained_attr, ret)
```

So next function is `_highest_gain_attr`, and it's role is to calculate *info* and *gain* value by tuples, and return highest gain attribute and divided dataset.

```
140     def _highest_gain_attr(self, head, tuples):
141         """ Calculate Info and Gain value by tuples, and return highest gain attr and divided dataset
142 
143         :param head: Attribute(class) set
144         :param tuples: Data set
145         :return: Tuple<highest_gain_attr, divided dataset>
146         """
147         cnt_table = {}
148         data_table = {}
```

Firstly, initialize cnt_table that includes counting information of each attribute.

```
150         for index, attr in enumerate(head):
151             if index == len(head) - 1:
152                 # Do not calculate result column
153                 break
154 
155             cnt_table[attr] = {}
156             data_table[attr] = {}
157 
158             if attr not in self.DOMAINS:
159                 print("Warning: Attribute '%s' is not able in this program" % attr)
160                 continue
161 
162             for domain in self.DOMAINS[attr]:
163                 cnt_table[attr][domain] = {}
164                 data_table[attr][domain] = []
165 
166                 for rst_domain in self.DOMAINS[head[-1]]:
167                     # Init table like `['age']['<=30']['yes']=0`
168                     cnt_table[attr][domain][rst_domain] = 0
169 
```

And then fill data on this table for calculating *info* and *gain* data in next step.

```
170         for tuple in tuples:
171             result_attr = tuple[-1]
172 
173             for index, data in enumerate(tuple):
174                 if index == len(tuple) - 1:
175                     # Ignore result column
176                     break
177 
178                 cnt_table[head[index]][data][result_attr] += 1
179                 data_table[head[index]][data].append(tuple)
180 
```

Finally calculate *info* data of each attribute by formula "Sigma( |Dj| * Info(Dj) ) / |D|", and choose min value among all attribute (candidates in this code).

```
182         candidates = []
183         for attr, D in cnt_table.items():
184             # Sigma( |Dj| * Info(Dj) ) / |D|
185             infoA = sum([
186                 sum(Dj.values()) * DecisionTree.info(*Dj.values()) for Dj in D.values()
187             ]) / len(tuples)
188 
189             candidates.append((attr, infoA))
190 
191         min_attr = min(candidates, key=lambda x: x[1])[0]
192 
193         return min_attr, data_table[min_attr]
194 
```

Info function on above formula is implemented in `info` function.

```
196     @staticmethod
197     def info(*arg):
198         """ Calculating Info(D) function
199 
200         :param *arg: List of tuple counts
201         :return: Ranged value from 0 to 1
202         """
203 
204         ret = 0
205         s = sum(arg)
206         if s == 0: return 0
207 
208         for a in arg:
209             p = a/s
210             if p == 0: continue
211             ret += -(p * math.log(p, 2))
212         return ret
```

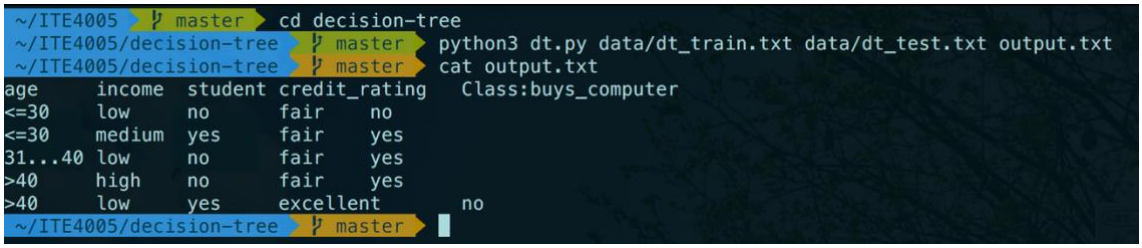
This is end of DecisionTree, and process for getting arguments, constructing class, and run test by files is implemented in the last of code.

```
215 if __name__ == '__main__':
216     if len(sys.argv) != 4:
217         print("Usage: python dt.py <training_set> <test_set> <output_file>")
218         sys.exit(-1)
219 
220     dt = DecisionTree(open(sys.argv[1], 'r'))
221     dt.testfile(
222         open(sys.argv[2], 'r'),
223         open(sys.argv[3], 'w'),
224     )
```

### 3) Instruction for compiling

Codes are written by python3, so please not to run with python2.

```
$ python3 dt.py data/dt_train.txt data/dt_test.txt output.txt
```




You can also run tests by below instruction. (pytest is required)