Decision Tree

- Programming assignment #2

2013011060
양승호

1. Environment

Ubuntu Linux 18.04
Python 3.6.7

2. How to run

$ python3 dt.py [train_file_name] [test_file_name] [result_file_name]
(example: python3 dt.py dt_train1.txt dt_test1.txt dt_result1.txt)

3. Algorithm Summary

- Decision Tree Constructing

1. Find an attribute of maximum information gain from entire db.
2. Set the root node of decision tree using the information gain.
3. As moving to child node, modify the db following the attribute of parent node and repeat from [Procedure 1] to [Procedure 2] using the modified db until there's neither samples nor attributes left, or all the remaining samples from the node are in same class.

4. Details

```
27 def InformationGain(dbs: list, attrs: list, attr_set: list, exceptions: list
   ):
28     gains = [0] * len(attrs)
29 
30     # Calculate each Information Gains
31     class_count = collections.defaultdict(lambda: 0)
32     for tup in dbs:
33         if tup[1]:
34             class_count[tup[0][-1]] += 1
35 
36     ## Calculate the original entropy
37     each_number = []
38     key_list = list(class_count)
39     total_number = 0
40     for key in key_list:
41         each_number.append(class_count[key])
42         total_number += class_count[key]
43     origin_entropy = 0
44     for i in range(len(each_number)):
45         each_number = class_count[key_list[i]]
46         each_prob = each_number / total_number
47         origin_entropy -= each_prob * math.log(each_prob, 2)
48 
49     ## Prepare to count each attribute values with class label
50     attr_count_list = []
51     for idx in range(len(attr_set)):#attr_group in attr_set:
52         attr_count = dict() # defaultdict and lambda initialize?
53         tmp = list(attr_set[idx])
54         for item in tmp:
55             tmp_dict = dict()
56             for key in key_list:
57                 tmp_dict[key] = 0
```

```
69      ## Calculate the divided entropy
70      for idx in range(len(gains)):
71          divided_entropy = 0
72          for col in attr_count_list[idx]:
73              cont_flag = False
74              if col[0] in exceptions:
75                  continue
76              if attrs[idx] == col[0]:
77                  local_each_number = []
78                  for key in key_list:
79                      local_label = attr_count_list[idx][col][key]
80                      if local_label == 0:
81                          cont_flag = True
82                          continue
83                      local_each_number.append(local_label)
84
85                  if cont_flag:
86                      continue
87                  else:
88                      local_total = sum(local_each_number)
89                      local_each_prob = []
90                      for number in local_each_number:
91                          prob = number / local_total
92                          local_each_prob.append(prob)
93                      for prob in local_each_prob:
94                          divided_entropy -= (local_total / total_number) * (prob * math.log(prob, 2
    ))
95
96          gains[idx] = origin_entropy - divided_entropy
97
98      # Exclude the class label column
99      gains[-1] = -100
00      for attr in exceptions:
01          gains[attrs.index(attr)] = -100
02      return gains, attr_count_list
```

- InformationGain : return the list of information gain of each attributes using the modified dbs following the parents' attributes.
- First, get the number of class labels of db and calculate the original entropy
- and then calculate the information gain of each attribute except for attributes already chosen.
- Exclude the attributes already chosen by setting them the minimum value not to be chosen again.

```python
def GainRatio(dbs: list, gains: list, attrs: list, attr_count_list: list):
    split_infos = [0] * len(attrs)

    class_count = collections.defaultdict(lambda: 0)
    for tup in dbs:
        if tup[1]:
            class_count[tup[0][-1]] += 1

    each_number = []
    key_list = list(class_count)
    total_number = 0
    for key in key_list:
        each_number.append(class_count[key])
        total_number += class_count[key]

    ## Calculate the numbers
    yes_number = class_count['yes']
    no_number = class_count['no']
    for idx in range(len(split_infos)):
        for col in attr_count_list[idx]:
            if col[0] == attrs[idx]:
                local_total = attr_count_list[idx][col]['yes'] + attr_count_list[idx][col]['no']
                if local_total == 0:
                    continue
                split_infos[idx] -= (local_total / total_number) * math.log(local_total / total_nu
mber, 2)

    gain_ratio = [0] * len(attrs)
    for idx in range(len(gains)):
        if split_infos[idx] == 0:
            continue
        gain_ratio[idx] = gains[idx]/split_infos[idx]
    # Exclude the class label column and exceptions
    gain_ratio[-1] = -100
    #for attr in exceptions:
    #   GainRatio[attrs.index(attr)] = -100
    return gain_ratio
```

- GainRatio : fx that return gain ratio for studying(not used in this project)

```python
def ConstructDT(dbs: list, attrs: list, dt: DecisionTree, exceptions: list, class_labels_count: dict):
    #
    class_count = collections.defaultdict(lambda: 0)
    for tup in dbs:
        if tup[1]:
            class_count[tup[0][-1]] += 1
    each_number = []
    key_list = list(class_count)
    for key in key_list:
        each_number.append(class_count[key])

    # DEBUGGING SECTION #
    #print('================================')
    #for tup in dbs:
    #   print(tup)
    #print(exceptions)
    #print('key list is', key_list)
    #print('each number is', each_number)


    # Termination Condition 1 : All samples belong to the same class OR there are no samples left
    if each_number.count(0) == len(each_number) - 1:
        dt.inherit = key_list[0]
        class_labels_count[dt.inherit] += 1
        return
    else:
    # Termination Condition 2 : All attributes are already considered to make current sub decision tree
        all_attr_in = True
        for attr in attrs[:-1]:
            if attr not in exceptions:
                all_attr_in = False
                break
        if all_attr_in:
            if each_number:
                dt.inherit = key_list[each_number.index(min(each_number))]

```

```
178             # Very Special Exception Case. . .
179             else:
180                 not_in = 0
181                 for key in list(class_labels_count):
182                     if class_labels_count[key] == 0:
183                         not_in = key
184                         break
185                 #print('UNKNOWN')
186                 dt.inherit = not_in
187                 class_labels_count[dt.inherit] += 1
188             return
189 
190         # Get the maximum Gain or Gain Ratio attribute
191         gains, attr_count_list = InformationGain(dbs, attributes, attr_set, exceptions)
192 
193         # DEBUGING SECTION #
194         #gain_ratio = GainRatio(dbs, gains, attributes, attr_count_list)
195         #print('gains', gains)
196         #print('gain_ratio', gain_ratio)
197         max_gain_idx = 0
198         for idx in range(1, len(gains)):
199             if attr[max_gain_idx] in exceptions:
200                 max_gain_idx = idx
201             if gains[max_gain_idx] < gains[idx] and attr[idx] not in exceptions:
202                 max_gain_idx = idx
203 
204         # Using Information Gain,
205         exceptions.append(attrs[max_gain_idx])
206         print(exceptions)
207         # If root,
208         if dt.parent == None:
209             dt.inherit = attrs[max_gain_idx]
210             class_labels_count[dt.inherit] += 1
211             for value in list(attr_set)[max_gain_idx]:
212                 child = DecisionTree(dt.inherit, value, None)
213                 dt.appendChild(child)
214             for child in dt.children:
215                 #print(max_gain_idx, child.attr)
216                 tmp_dbs = ModifyDB(dbs, max_gain_idx, child.attr)
217                 exceptions_tmp = exceptions.copy()
218                 class_labels_count_tmp = class_labels_count.copy()
```

- ConstructDT: Construct the Decision Tree using information gain
- if there are neither samples nor attributes or all the samples belong to same class or over the half of them does, then stop constructing the sub decision tree(Termination Condition)
- else, recursively keep constructing

```
214             for child in dt.children:
215                 #print(max_gain_idx, child.attr)
216                 tmp_dbs = ModifyDB(dbs, max_gain_idx, child.attr)
217                 exceptions_tmp = exceptions.copy()
218                 class_labels_count_tmp = class_labels_count.copy()
219                 ConstructDT(tmp_dbs, attrs, child, exceptions_tmp, class_labels_count_tmp)
220             # Add child node
221             # And iterate the children using recursive
222         else:
223             dt.inherit = attrs[max_gain_idx]
224             for value in list(attr_set)[max_gain_idx]:
225                 child = DecisionTree(dt.inherit, value, None)
226                 dt.appendChild(child)
227             for child in dt.children:
228                 #print(max_gain_idx, child.attr)
229                 tmp_dbs = ModifyDB(dbs, max_gain_idx, child.attr)
230                 exceptions_tmp = exceptions.copy()
231                 class_labels_count_tmp = class_labels_count.copy()
232                 ConstructDT(tmp_dbs, attrs, child, exceptions_tmp, class_labels_count_tmp)
233 
234         return dt
235 
236 # Collect datas which  have specific attribute value
237 def ModifyDB(dbs: list, attr_idx: int, value: str):
238     ret = dbs.copy()
239     for idx in range(len(ret)):#tup in ret:
240         if ret[idx][0][attr_idx] != value:
241             ret[idx] = (ret[idx][0], False)
242     return ret
243 
244 def DetermineClass(tx: list, attrs: list, dt: DecisionTree):
245     if not dt.children:
246         #print(dt.inherit)
247         return dt.inherit
248     else:
249         for child in dt.children:
250             if tx[attrs.index(dt.inherit)] == child.attr:
251                 return DetermineClass(tx, attrs, child)
```

- ModifyDB: modify the dbs using specific attributes not to make corresponding datas to be considerted to Decision Tree making.
- set True, False to each samples

- DetermineClass: Determine the class of test samples using decision tree already made.

```
 6 class DecisionTree:
 7     def __init__(self, parent_attr, value, inheritance):
 8         self.parent = parent_attr
 9         self.attr = value
10         self.inherit = inheritance
11         self.children = []
12 
13     def appendChild(self, child):
14         self.children.append(child)
15 
16     def printTree(self, depth: int):
17         print(depth)
18         print('Parent is', self.parent)
19         print('Attribute is', self.attr)
20         print('Inheritance is', self.inherit)
21         if self.children:
22             print('Children is')
23             for child in self.children:
24                 child.printTree(depth+1)
```

- Decision Tree data structure
- parent stands for the attribute of parent, which current attr belongs to
- attr stands for the vale of parent
- inherit stnads for the attribute of children
- children is a list of child nodes of current node
- printTree: for debugging, check the decision tree made.

```
274     class_labels = list(attr_set[-1])
275     class_labels_count = collections.defaultdict(lambda: 0)
276     for label in class_labels:
277         class_labels_count[label] = 0
278     exceptions = []
279 
280     dt = DecisionTree(None, None, None)
281     ConstructDT(dbs, attributes, dt, exceptions, class_labels_count)
282     # Print the whole Decision Tree
283     #dt.printTree(0)
284 
285     test_file = open(test_file_name, 'r')
286     test_lines = test_file.readlines()
287     test_dbs = []
288     for line in test_lines[1:]:
289         tx = line.split()
290         test_dbs.append(tx)
291 
292     output_file = open(output_file_name, 'w')
293     for attr in attributes:
294         output_file.write(attr + '\t')
295     output_file.write('\n')
296     for tx in test_dbs:
297         for value in tx:
298             output_file.write(value + '\t')
299         output_file.write(DetermineClass(tx, attributes, dt))
300         output_file.write('\n')
```

- main function

5. Testing Result(result1.txt) T^T