**智能信息系统综合实践**

**实验报告**

|  |  |
| --- | --- |
| **题 目：** | 决策树 |
| **年 级：** | **2020级** |
| **专 业：** | **软件工程** |
| **姓 名：** | **庞晓宇** |

1. **题目（原题目）**

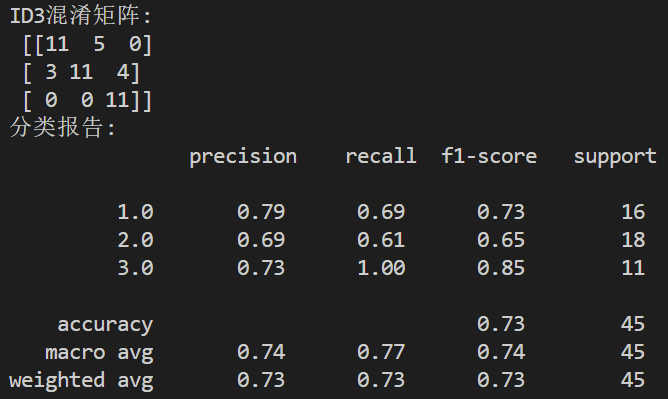
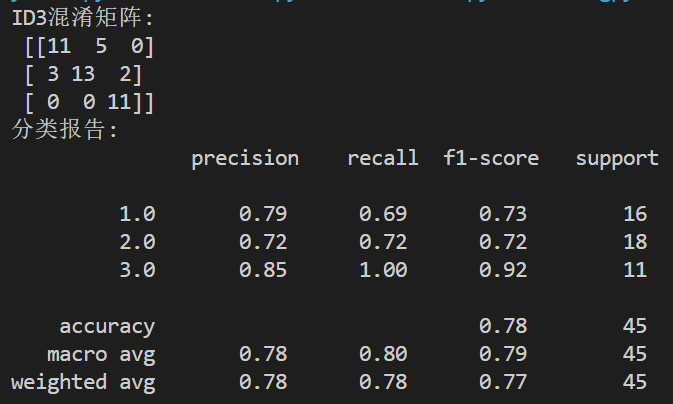
根据软木塞数据集，利用C4.5算法（不能调包）生成决策树模型。

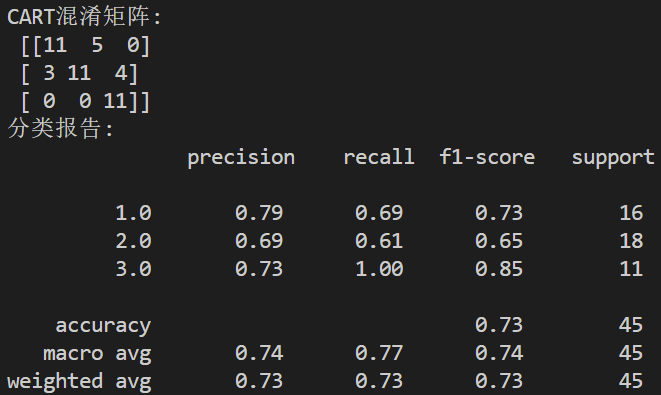
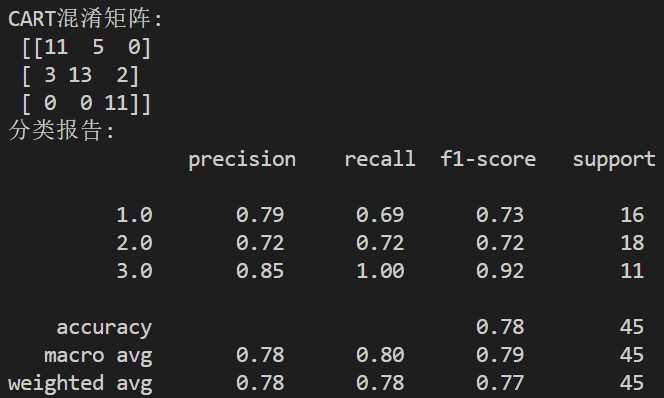
要求：1. 随机选取训练集和测试集

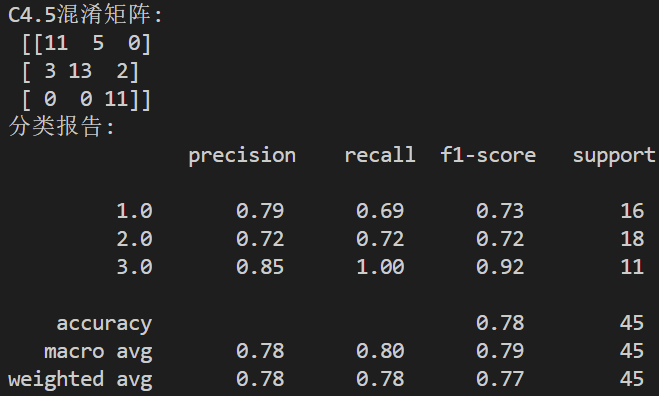
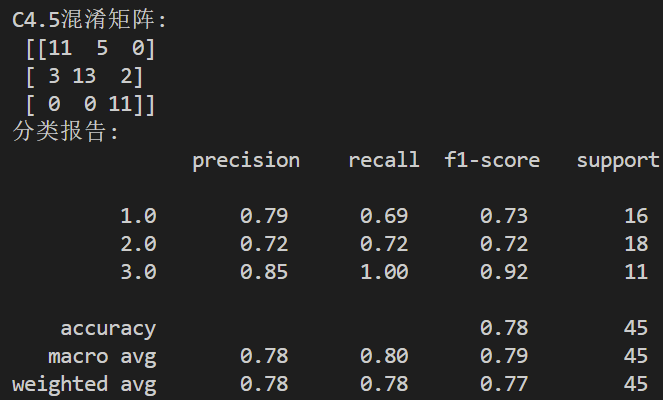
2. 生成决策树模型，并对模型进行评估（混淆矩阵，查全率，查准率F1值）

3. 使用CART算法（可调包）生成决策树模型与C4.5算法结果对比，并评价这两种算法的优缺点。

1. **解题步骤（思路+代码）**
2. import collections
3. import operator
4. from math import log
5. import pandas as pd
6. import numpy as np
7. from sklearn.metrics import classification\_report, confusion\_matrix
8. from sklearn.model\_selection import train\_test\_split
9. from sklearn.tree import DecisionTreeClassifier
10. # 调试
11. def pp(d):
12. import pprint
13. pprint.pprint(d)
14. exit(0)
15. # 将训练集转换为列表形式
16. def create\_dataset(X\_train, y\_train):
17. dataset = []
18. for i in range(len(X\_train)):
19. dataset.append(list(X\_train.iloc[i]) + [int(y\_train.iloc[i])])
20. # 特征值列表
21. labels = list(X\_train.columns)
22. # 特征对应的所有可能的情况
23. labels\_full = {}
24. for i in range(len(labels)):
25. label\_list = [example[i] for example in dataset]
26. unique\_label = set(label\_list)
27. labels\_full[labels[i]] = unique\_label
28. return dataset, labels, labels\_full
29. # 找到次数最多的类别标签
30. def majority\_count(class\_list):
31. *# 用来统计标签的票数*
32. class\_count = collections.defaultdict(int)
33. # 遍历所有的标签类别
34. for vote in class\_list:
35. class\_count[vote] += 1
36. # 从大到小排序
37. sorted\_class\_count = sorted(
38. class\_count.items(), key=operator.itemgetter(1), reverse=True)
39. # 返回次数最多的标签
40. return sorted\_class\_count[0][0]
41. # 计算给定数据集的信息熵(香农熵)
42. def calc\_shannon\_entropy(dataset):
43. *# 计算出数据集的总数*
44. entries\_number = len(dataset)
45. # 用来统计标签
46. label\_counts = collections.defaultdict(int)
47. # 循环整个数据集，得到数据的分类标签
48. for feature\_vector in dataset:
49. *# 得到当前的标签*
50. current\_label = feature\_vector[-1]
51. # 将对应的标签值加一
52. label\_counts[current\_label] += 1
53. # 默认的信息熵
54. shannon\_entropy = 0.0
55. for key in label\_counts:
56. *# 计算出当前分类标签占总标签的比例数*
57. prob = float(label\_counts[key]) / entries\_number
58. # 以2为底求对数
59. shannon\_entropy -= prob \* log(prob, 2)
60. return shannon\_entropy
61. # 按照给定的数值，将数据集分为不大于和大于两部分
62. def split\_dataset4series(dataset, axis, value):
63. *# 用来保存不大于划分值的集合*
64. elt\_dataset = []
65. *# 用来保存大于划分值的集合*
66. gt\_dataset = []
67. *# 进行划分，保留该特征值*
68. for feat in dataset:
69. if feat[axis] <= value:
70. elt\_dataset.append(feat)
71. else:
72. gt\_dataset.append(feat)
73. return elt\_dataset, gt\_dataset
74. # 按照给定的特征值，将数据集划分
75. def split\_dataset(dataset, axis, value):
76. *# 创建一个新的列表，防止对原来的列表进行修改*
77. ret\_dataset = []
78. # 遍历整个数据集
79. for feature\_vector in dataset:
80. *# 如果给定特征值等于想要的特征值*
81. if feature\_vector[axis] == value:
82. *# 将该特征值前面的内容保存起来*
83. reduced\_feature\_vector = feature\_vector[:axis]
84. *# 将该特征值后面的内容保存起来，所以将给定特征值给去掉了*
85. reduced\_feature\_vector.extend(feature\_vector[axis + 1:])
86. *# 添加到返回列表中*
87. ret\_dataset.append(reduced\_feature\_vector)
88. return ret\_dataset
89. # 计算连续值的信息增益
90. def calc\_infogain4series(dataset, index, base\_entropy):
91. *# 记录最大的信息增益*
92. max\_infogain = 0.0
93. # 最好的划分点
94. best\_mid = -1
95. # 得到数据集中所有的当前特征值列表
96. feature\_list = [example[index] for example in dataset]
97. # 得到分类列表
98. class\_list = [example[-1] for example in dataset]
99. dict\_list = dict(zip(feature\_list, class\_list))
100. # 将其从小到大排序，按照连续值的大小排列
101. sorted\_feature\_list = sorted(dict\_list.items(), key=operator.itemgetter(0))
102. # 计算连续值有多少个
103. feature\_list\_number = len(sorted\_feature\_list)
104. # 计算划分点，保留三位小数
105. mid\_feature\_list = [round((sorted\_feature\_list[i][0] + sorted\_feature\_list[i+1][0])/2.0, 3)
106. for i in range(feature\_list\_number - 1)]
107. # 计算出各个划分点信息增益
108. for mid in mid\_feature\_list:
109. *# 将连续值划分为不大于当前划分点和大于当前划分点两部分*
110. elt\_dataset, gt\_dataset = split\_dataset4series(dataset, index, mid)
111. # 计算两部分的特征值熵和权重的乘积之和
112. new\_entropy = len(elt\_dataset)/len(sorted\_feature\_list)\*calc\_shannon\_entropy(
113. elt\_dataset) + len(gt\_dataset)/len(sorted\_feature\_list)\*calc\_shannon\_entropy(gt\_dataset)
114. # 计算出信息增益
115. infogain = base\_entropy - new\_entropy
116. *# 计算信息增益率*
117. infogain = (base\_entropy - new\_entropy) / base\_entropy
118. if infogain > max\_infogain:
119. best\_mid = mid
120. max\_infogain = infogain
121. return max\_infogain, best\_mid
122. # 计算信息增益
123. def calc\_infogain(dataset, feature\_list, current\_index, base\_entropy):
124. *# 将当前特征唯一化，也就是说当前特征值中共有多少种*
125. unique\_values = set(feature\_list)
126. # 新的熵，代表当前特征值的熵
127. new\_entropy = 0.0
128. # 遍历现在有的特征的可能性
129. for value in unique\_values:
130. *# 在全部数据集的当前特征位置上，找到该特征值等于当前值的集合*
131. sub\_dataset = split\_dataset(
132. dataset=dataset, axis=current\_index, value=value)
133. *# 计算出权重*
134. prob = len(sub\_dataset) / float(len(dataset))
135. *# 计算出当前特征值的熵*
136. new\_entropy += prob \* calc\_shannon\_entropy(sub\_dataset)
137. # 计算出“信息增益”
138. infogain = base\_entropy - new\_entropy
139. *# 计算信息增益率*
140. infogain = (base\_entropy - new\_entropy) / base\_entropy
141. return infogain
142. # 选择最好的数据集划分特征，根据信息增益值来计算，可处理连续值
143. def choose\_best\_splitfeature(dataset, labels):
144. *# 得到数据的特征值总数*
145. features\_number = len(dataset[0]) - 1
146. # 计算出基础信息熵
147. base\_entropy = calc\_shannon\_entropy(dataset)
148. # 基础信息增益为0.0
149. best\_infogain = 0.0
150. # 最好的特征值
151. best\_feature = -1
152. # 标记当前最好的特征值是不是连续值
153. flag\_series = 0
154. # 如果是连续值的话，用来记录连续值的划分点
155. best\_series\_mid = 0.0
156. # 对每个特征值进行求信息熵
157. for i in range(features\_number):
158. # 得到数据集中所有的当前特征值列表
159. feature\_list = [example[i] for example in dataset]
160. if isinstance(feature\_list[0], str):
161. infogain = calc\_infogain(dataset, feature\_list, i, base\_entropy)
162. else:
163. *# print('当前划分属性为：' + str(labels[i]))*
164. infogain, best\_mid = calc\_infogain4series(
165. dataset, i, base\_entropy)
166. # 如果当前的信息增益比原来的大
167. if infogain > best\_infogain:
168. *# 最好的信息增益*
169. best\_infogain = infogain
170. *# 新的最好的用来划分的特征值*
171. best\_feature = i
172. flag\_series = 0
173. if not isinstance(dataset[0][best\_feature], str):
174. flag\_series = 1
175. best\_series\_mid = best\_mid
176. if flag\_series:
177. return best\_feature, best\_series\_mid
178. else:
179. return best\_feature
180. # 创建决策树
181. def create\_tree(dataset, labels):
182. *# 拿到所有数据集的分类标签*
183. class\_list = [example[-1] for example in dataset]
184. # 统计第一个标签出现的次数，与总标签个数比较，如果相等则说明当前列表中全部都是一种标签，此时停止划分
185. if class\_list.count(class\_list[0]) == len(class\_list):
186. return class\_list[0]
187. # 计算第一行有多少个数据，如果只有一个的话说明所有的特征属性都遍历完了，剩下的一个就是类别标签
188. if len(dataset[0]) == 1:
189. *# 返回剩下标签中出现次数较多的那个*
190. return majority\_count(class\_list)
191. # 选择最好的划分特征，得到该特征的下标
192. best\_feature = choose\_best\_splitfeature(dataset=dataset, labels=labels)
193. # 得到最好特征的名称
194. best\_feature\_label = ''
195. # 记录此刻是连续值还是离散值,1连续，2离散
196. flag\_series = 0
197. # 如果是连续值，记录连续值的划分点
198. mid\_series = 0.0
199. # 如果是元组的话，说明此时是连续值
200. if isinstance(best\_feature, tuple):
201. *# 重新修改分叉点信息*
202. best\_feature\_label = str(labels[best\_feature[0]]) + \
203. '小于' + str(best\_feature[1]) + '?'
204. *# 得到当前的划分点*
205. mid\_series = best\_feature[1]
206. *# 得到下标值*
207. best\_feature = best\_feature[0]
208. *# 连续值标志*
209. flag\_series = 1
210. else:
211. *# 得到分叉点信息*
212. best\_feature\_label = labels[best\_feature]
213. *# 离散值标志*
214. flag\_series = 0
215. # 使用一个字典来存储树结构，分叉处为划分的特征名称
216. my\_tree = {best\_feature\_label: {}}
217. # 得到当前特征标签的所有可能值
218. feature\_values = [example[best\_feature] for example in dataset]
219. # 连续值处理
220. if flag\_series:
221. *# 将连续值划分为不大于当前划分点和大于当前划分点两部分*
222. elt\_dataset, gt\_dataset = split\_dataset4series(
223. dataset, best\_feature, mid\_series)
224. *# 得到剩下的特征标签*
225. sub\_labels = labels[:]
226. *# 递归处理小于划分点的子树*
227. sub\_tree = create\_tree(elt\_dataset, sub\_labels)
228. my\_tree[best\_feature\_label]['小于'] = sub\_tree
229. # 递归处理大于当前划分点的子树
230. sub\_tree = create\_tree(gt\_dataset, sub\_labels)
231. my\_tree[best\_feature\_label]['大于'] = sub\_tree
232. return my\_tree
233. # 离散值处理
234. else:
235. *# 将本次划分的特征值从列表中删除掉*
236. del (labels[best\_feature])
237. *# 唯一化，去掉重复的特征值*
238. unique\_values = set(feature\_values)
239. *# 遍历所有的特征值*
240. for value in unique\_values:
241. *# 得到剩下的特征标签*
242. sub\_labels = labels[:]
243. *# 递归调用，将数据集中该特征等于当前特征值的所有数据划分到当前节点下，递归调用时需要先将当前的特征去除掉*
244. sub\_tree = create\_tree(split\_dataset(
245. dataset=dataset, axis=best\_feature, value=value), sub\_labels)
246. *# 将子树归到分叉处下*
247. my\_tree[best\_feature\_label][value] = sub\_tree
248. return my\_tree
249. def mypredict(input\_tree, feature\_values):
250. *# 得到树的根节点*
251. first\_str = list(input\_tree.keys())[0]
252. # 得到根节点的所有子节点
253. second\_dict = input\_tree[first\_str]
254. # 得到根节点的特征标签
255. feature\_label = first\_str.split('小于')[0]
256. # 得到根节点的划分点
257. mid\_series = float(first\_str.split('小于')[1].split('?')[0])
258. # 得到根节点的特征值
259. feature\_value = feature\_values[feature\_label]
260. # 如果是连续值
261. if isinstance(feature\_value, float) or isinstance(feature\_value, int):
262. *# 如果当前特征值小于等于划分点*
263. if feature\_value <= mid\_series:
264. *# 得到小于划分点的子树*
265. value\_of\_feat = second\_dict['小于']
266. else:
267. *# 得到大于划分点的子树*
268. value\_of\_feat = second\_dict['大于']
269. else:
270. *# 得到当前特征值对应的子树*
271. value\_of\_feat = second\_dict[feature\_value]
272. # 如果子树是一个字典的话，说明还没有到叶子节点，继续递归调用
273. if isinstance(value\_of\_feat, dict):
274. class\_label = mypredict(value\_of\_feat, feature\_values)
275. else:
276. *# 如果子树是一个字符串的话，说明已经到了叶子节点，直接返回当前的类别标签*
277. class\_label = value\_of\_feat
278. return class\_label
279. def mypredicts(input\_tree, X\_test, labels):
280. *# 得到测试集的行数*
281. rows = X\_test.shape[0]
282. *# 存储预测结果*
283. y\_pred = []
284. *# 遍历每一行*
285. for i in range(rows):
286. *# 得到当前行的特征值，结合labels，转换为字典格式*
287. feature\_values = dict(zip(labels, X\_test.iloc[i, :]))
288. *# 得到当前行的预测结果*
289. y\_pred.append(mypredict(input\_tree, feature\_values))
290. *# 将预测结果转换为ndarray格式*
291. y\_pred = np.array(y\_pred)
292. return y\_pred
293. if \_\_name\_\_ == '\_\_main\_\_':
294. data = pd.read\_excel('DATASETS\_CorkStoppers.xls', sheet\_name='Data')
295. data.head()
296. # 数据预处理
297. data = data.dropna()  # 删除空值
298. data = data.drop\_duplicates()  # 删除重复值
299. data = data.drop(['#'], axis=1)
300. data.head()
301. # 划分训练集和测试集
302. y = data['C']
303. X = data.drop(['C'], axis=1)
304. X\_train, X\_test, y\_train, y\_test = train\_test\_split(
305. X, y, test\_size=0.3, random\_state=0)
306. # ID3决策树模型
307. id3 = DecisionTreeClassifier(criterion='entropy')
308. id3.fit(X\_train, y\_train)
309. y\_pred\_id3 = id3.predict(X\_test)
310. print('ID3混淆矩阵:\n', confusion\_matrix(y\_test, y\_pred\_id3))
311. print('分类报告:\n', classification\_report(y\_test, y\_pred\_id3))
312. # 使用CART算法（可调包）生成决策树模型
313. cart = DecisionTreeClassifier()
314. cart.fit(X\_train, y\_train)
315. y\_pred\_cart = cart.predict(X\_test)
316. print('CART混淆矩阵:\n', confusion\_matrix(y\_test, y\_pred\_cart))
317. print('分类报告:\n', classification\_report(y\_test, y\_pred\_cart))
318. # 使用C4.5算法生成决策树模型
319. dataset, labels, labels\_full = create\_dataset(X\_train, y\_train)
320. c45 = create\_tree(dataset, labels)
321. y\_pred\_c45 = mypredicts(c45, X\_test, labels)
322. *# print(y\_pred\_c45)*
323. *# pp(y\_pred\_c45)*
324. print('C4.5混淆矩阵:\n', confusion\_matrix(y\_test, y\_pred\_c45))
325. print('分类报告:\n', classification\_report(y\_test, y\_pred\_c45))

…

…

…

1. **总结（心得体会）**

决策树算法能够处理多种类型的数据，易于理解和解释。常用的决策树算法有ID3算法、C4.5算法和CART算法。ID3算法简单易实现，但对于取值数目较多的特征容易过拟合；C4.5算法可以处理连续值和离散值，相对于ID3算法减少了过拟合的情况，但时间复杂度较高；CART算法可以处理连续值和离散值，相对于ID3算法减少了过拟合的情况，但只能生成二叉树，对于样本数据分布不均衡的情况容易产生偏差。应根据实际问题选择合适的算法，数据量较小时选择ID3算法，数据量较大时选择C4.5算法和CART算法。

通过多次对原始数据集的划分和训练，发现各个算法的准确度差别并不显著。因为属性都是连续取值，也就是说在划分数据集的时候只会划分为两个属性集，而C4.5算法本身是为了避免ID3对于取值较多属性的偏好，这里的优势似乎体现不出来，虽说计算信息增益率似乎也并没有很影响训练性能。