

一、绘制一个棋盘和求解水仙花问题

```
6 import sys
7
8 for i in range(8):
9     for j in range(8):
10         if (i+j)%2 == 0:
11             sys.stdout.write(chr(219))
12             sys.stdout.write(chr(219))
13         else:
14             sys.stdout.write(' ')
15     print (' ')
```

```
49 #水仙花问题
50 def jude(n):
51     m=n
52     a=int(m/100)
53     m=m%100
54     b=int(m/10)
55     m=m%10
56     if pow(m,3)+pow(a,3)+pow(b,3)==n:
57         return True
58     else:
59         return False
60
61 for i in range(100,1000):
62     if jude(i)==True and i!=407:
63         print(i,end=',')
64     elif i==407:
65         print(i)
```

二、python 开发的学生信息管理系统

```
2 def showInfo():
3     """
4     显示可以使用的功能列表给用户
5     """
6     print("-"*30)
7     print("      学生管理系统      ")
8     print(" 1.添加学生的信息")
9     print(" 2.删除学生的信息")
10    print(" 3.修改学生的信息")
11    print(" 4.查询学生的信息")
12    print(" 5.遍历所有学生的信息")
13    print(" 0.退出系统")
14    print('-'*30)
15
16 #定义一个列表，用来存储多个学生的信息
17 students=[]
```

```
20 def addStudent():
21     '''
22         添加一个学生，需要传入姓名、年龄、学号
23     '''
24     #输入学员姓名、年龄、学号
25     stuName = input("请输入学生姓名: ")
26     stuId = input("请输入学生学号(学号不可重复): ")
27     stuAge = input("请输入学生年龄:")
28     #验证学号是否唯一 #i记录要删除的下标，Leap为标志位，如果找到Leap=1，否则为0
29     i = 0
30     leap = 0
31     #循环判断
32     for stu in students:
33         if stu['stuId'] == stuId:
34             leap = 1
35             break
36         else:
37             i = i + 1
38     #leap == 1代表学生学号
39     if leap == 1:
40         print("输入学生学号重复，添加失败!")
41     else:
42         # 定义一个字典，存放单个学生信息
43         stuInfo = {}
44         stuInfo['stuName'] = stuName
45         stuInfo['stuId'] = stuId
46         stuInfo['stuAge'] = stuAge
47
48         # 单个学生信息放入列表
49         students.append(stuInfo)
50         print("添加成功!")
```

```
52 #删除学生函数
53 def deleteStudent():
54     '''
55         根据学号删除学生，学号
56     '''
57     print("您选择了删除学生功能")
58     delId=input("请输入要删除的学生学号:")
59     #i记录要删除的下标，Leap为标志位，如果找到Leap=1，否则为0
60     i = 0
61     leap = 0
62     for stu in students:
63         if stu['stuId'] == delId:
64             leap = 1
65             break
66         else:
67             i=i+1
68     if leap == 0:
69         print("没有此学生学号，删除失败!")
70     else:
71         del students[i]
72         print("删除成功!")
```

```
75 #修改学生函数, 有bug
76 def updateStudent():
77     '''
78     根据学号修改学生信息, 学号
79     '''
80     print("您选择了修改学生信息功能")
81     alterId=input("请输入你要修改学生的学号:")
82     #检测是否有此学号, 然后进行修改信息
83     i = 0
84     leap = 0
85     for stu in students:
86         if stu['stuId'] == alterId:
87             leap = 1
88             break
89         else:
90             i = i + 1
91     if leap == 1:
92         updateOperate()
93     else:
94         print("没有此学号, 修改失败! ")

96 def updateOperate():
97     '''
98     根据用户选择不同的操作来修改学生的信息
99     '''
100     while True:
101         alterNum=int(input(" 1.修改学号\n 2.修改姓名 \n 3.修改年龄 \n 4.退出修改\n"))
102         if alterNum == 1:
103             newId=input("输入更改后的学号:")
104             #修改后的学号要验证是否唯一
105             i = 0
106             leap1 = 0
107             for stu1 in students:
108                 if stu1['stuId'] == newId:
109                     leap1 = 1
110                     break
111                 else:
112                     i = i + 1
113             if leap1 == 1:
114                 print("输入学号不可重复, 修改失败! ")
115             else:
116                 stu1['stuId']=newId
117                 print("学号修改成功")
118             elif alterNum == 2: #修改姓名操作
119                 newName=input("输入更改后的姓名:")
120                 stu['stuName'] = newName
121                 print("姓名修改成功")

122             elif alterNum == 3: #修改年龄操作
123                 newAge=input("输入更改后的年龄:")
124                 stu['stuAge'] = newAge
125                 print("年龄修改成功")
126             elif alterNum == 4:
127                 break
128             else:
129                 print("输入错误请重新输入")
```

```

132 #查询单个学生信息函数
133 def getStudentById():
134     '''
135     根据学号查询学生信息,需要传入学号
136     '''
137     print("您选择了查询学生信息功能")
138     searchID=input("请输入你要查询学生的学号:")
139     #验证是否有此学号
140     i = 0
141     leap = 0
142     for stu in students:
143         if stu['stuId'] == searchID:
144             leap = 1
145             break
146         else:
147             i = i + 1
148     if leap == 0:
149         print("没有此学生学号, 查询失败!")
150     else:
151         print("找到此学生, 信息如下: ")
152         print("学号: %s\n姓名: %s\n年龄: %s\n"%(stu['stuId'],stu['stuName'],stu['stuAge']))

154 #查询所有学生信息函数
155 def getAllStudent():
156     '''
157     查询所有学生信息
158     '''
159     #遍历并输出所有学生的信息
160     print('*'*20)
161     print("接下来进行遍历所有的学生信息...")
162     print("stuId      姓名      年龄")
163     for stu in students:
164         print("%s      %s      %s"%(stu['stuId'],stu['stuName'],stu['stuAge']))
165     print('*'*20)

167 #主函数
168 def main():
169     '''
170     主函数: 程序的入口
171     '''
172     while True:
173         #把功能列表进行显示给用户
174         showInfo()
175
176         #提示用户选择功能
177         #获取用户选择的功能
178         key = int(input("请选择功能 (序号): "))
179
180         #根据用户选择, 完成相应功能
181         if key == 1:
182             addStudent()
183         elif key == 2:
184             deleteStudent()
185         elif key == 3:
186             updateStudent()
187         elif key == 4:
188             getStudentById()
189         elif key == 5:
190             getAllStudent()
191         elif key == 0:
192             #退出功能, 尽量往不退出的方向引
193             quitconfirm = input("亲, 真的要退出么 (yes或者no) ")
194             if quitconfirm == 'yes':
195                 print("欢迎使用本系统, 谢谢")
196                 break;1
197             else:
198                 print("您输入有误, 请重新输入")
199
200 main()

```

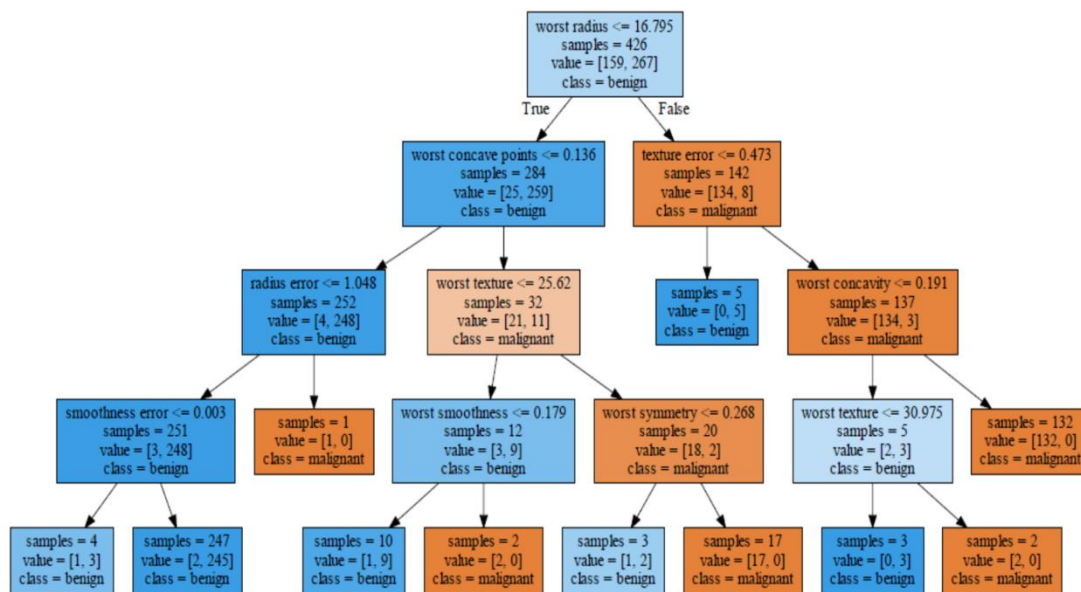
三、调用 sklearn 库和癌症数据集实现决策树

由于要使用 graphviz 函数，需要配置这个库，方法如下：

- 1) 打开 Anaconda Prompt 后，输入 `conda install graphviz`，然后再输入 `pip install graphviz`，
- 2) 将 graphviz 路径添加在系统环境变量中，如 `D:\Anaconda3\Library\bin\graphviz`。

```
6 from sklearn.tree import DecisionTreeClassifier
7 from sklearn.datasets import load_breast_cancer
8 from sklearn.model_selection import train_test_split
9
10 cancer = load_breast_cancer()
11 X_train, X_test, y_train, y_test = train_test_split(cancer.data,
12                                                    cancer.target,
13                                                    stratify=cancer.target,
14                                                    random_state=42)
15 tree = DecisionTreeClassifier(random_state=0)
16 tree.fit(X_train, y_train)
17 print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
18 print("Accuracy on testing set: {:.3f}".format(tree.score(X_test, y_test)))
19
20 tree = DecisionTreeClassifier(max_depth=4, random_state=0)
21 tree.fit(X_train, y_train)
22 print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
23 print("Accuracy on testing set: {:.3f}".format(tree.score(X_test, y_test)))
24
25
26 from sklearn.tree import export_graphviz
27
28 export_graphviz(tree,
29                 out_file="tree.dot",
30                 class_names=["malignant", "benign"],
31                 feature_names=cancer.feature_names,
32                 impurity=False,
33                 filled=True)
34
35 import graphviz
36 with open("tree.dot") as f:
37     dot_graph = f.read()
38
39 dot = graphviz.Source(dot_graph)
40 dot.view()
```

运行效果：



四、调用 sklearn 库和 iris 数据集实现决策树

```

6 import numpy as np
7 import pandas as pd
8 #import matplotlib.pyplot as plt
9 from sklearn.datasets import load_iris
10 from sklearn.model_selection import train_test_split
11 from sklearn.tree import DecisionTreeClassifier
12 from sklearn.tree import export_graphviz
13 import graphviz
14
15 # data
16 def create_data():
17     iris = load_iris()
18     df = pd.DataFrame(iris.data, columns=iris.feature_names)
19     df['label'] = iris.target
20     df.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'label']
21     data = np.array(df.iloc[:100, [0, 1, -1]])
22     # print(data)
23     return data[:, :2], data[:, -1]
24
25 X, y = create_data()
26 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
27

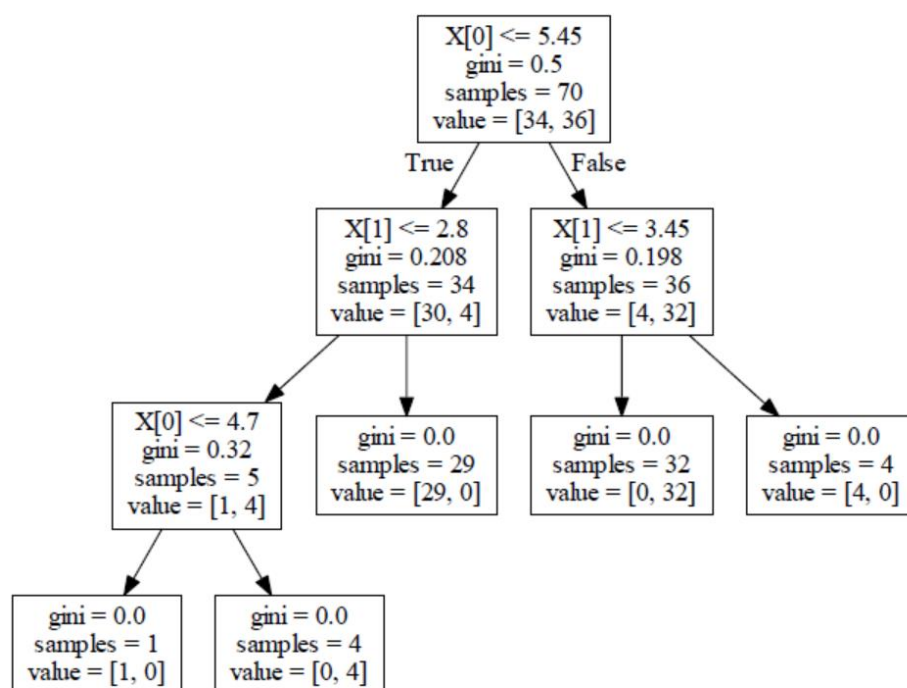
```

```

28 '''
29 DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
30                        max_features=None, max_leaf_nodes=None,
31                        min_impurity_decrease=0.0, min_impurity_split=None,
32                        min_samples_leaf=1, min_samples_split=2,
33                        min_weight_fraction_leaf=0.0, presort=False, random_state=None,
34                        splitter='best')
35 '''
36 clf = DecisionTreeClassifier()
37 clf.fit(X_train, y_train,)
38
39 clf.score(X_test, y_test)
40
41 tree_pic = export_graphviz(clf, out_file="mytree.pdf")
42 with open('mytree.pdf') as f:
43     dot_graph = f.read()
44
45 dot = graphviz.Source(dot_graph)
46 dot.view()
47

```

运行效果：



五、课本上的数据集

```

6 import numpy as np
7 import pandas as pd
8 from math import log
9 from sklearn.tree import export_graphviz
10
11 def create_data():
12     datasets = [['青年', '否', '否', '一般', '否'],
13                 ['青年', '否', '否', '好', '否'],
14                 ['青年', '是', '否', '好', '是'],
15                 ['青年', '是', '是', '一般', '是'],
16                 ['青年', '否', '否', '一般', '否'],
17                 ['中年', '否', '否', '一般', '否'],
18                 ['中年', '否', '否', '好', '否'],
19                 ['中年', '是', '是', '好', '是'],
20                 ['中年', '否', '是', '非常好', '是'],
21                 ['中年', '否', '是', '非常好', '是'],
22                 ['老年', '否', '是', '非常好', '是'],
23                 ['老年', '否', '是', '好', '是'],
24                 ['老年', '是', '否', '好', '是'],
25                 ['老年', '是', '否', '非常好', '是'],
26                 ['老年', '否', '否', '一般', '否']]
27     labels = [u'年龄', u'有工作', u'有自己的房子', u'信贷情况', u'类别']
28     # 返回数据集和每个维度的名称
29     return datasets, labels
30
31 datasets, labels = create_data()
32
33 train_data = pd.DataFrame(datasets, columns=labels)
34
35 #print (train_data)
36
37 # 熵
38 def calc_ent(datasets):
39     data_length = len(datasets)
40     label_count = {}
41     for i in range(data_length):
42         label = datasets[i][-1]
43         if label not in label_count:
44             label_count[label] = 0
45         label_count[label] += 1
46     ent = -sum([(p/data_length)*log(p/data_length, 2) for p in label_count.values()])
47     return ent
48
49 # 经验条件熵
50 def cond_ent(datasets, axis=0):
51     data_length = len(datasets)
52     feature_sets = {}
53     for i in range(data_length):
54         feature = datasets[i][axis]
55         if feature not in feature_sets:
56             feature_sets[feature] = []
57         feature_sets[feature].append(datasets[i])
58     cond_ent = sum([(len(p)/data_length)*calc_ent(p) for p in feature_sets.values()])
59     return cond_ent

```



```

61 # 信息增益
62 def info_gain(ent, cond_ent):
63     return ent - cond_ent
64
65 def info_gain_train(datasets):
66     count = len(datasets[0]) - 1
67     ent = calc_ent(datasets)
68     best_feature = []
69     for c in range(count):
70         c_info_gain = info_gain(ent, cond_ent(datasets, axis=c))
71         best_feature.append((c, c_info_gain))
72         print('特征({}) - info_gain - {:.3f}'.format(labels[c], c_info_gain))
73     # 比较大小
74     best_ = max(best_feature, key=lambda x: x[-1])
75     return '特征({})的信息增益最大, 选择为根节点特征'.format(labels[best_[0]])
76
77 info_gain_train(np.array(datasets))

80 # 定义节点类 二叉树
81 class Node:
82     def __init__(self, root=True, label=None, feature_name=None, feature=None):
83         self.root = root
84         self.label = label
85         self.feature_name = feature_name
86         self.feature = feature
87         self.tree = {}
88         self.result = {'label': self.label, 'feature': self.feature, 'tree': self.tree}
89
90     def __repr__(self):
91         return '{}'.format(self.result)
92
93     def add_node(self, val, node):
94         self.tree[val] = node
95
96     def predict(self, features):
97         if self.root is True:
98             return self.label
99         return self.tree[features[self.feature]].predict(features)
100
101 class DTree:
102     def __init__(self, epsilon=0.1):
103         self.epsilon = epsilon
104         self._tree = {}
105
106     # 熵
107     @staticmethod
108     def calc_ent(datasets):
109         data_length = len(datasets)
110         label_count = {}
111         for i in range(data_length):
112             label = datasets[i][-1]
113             if label not in label_count:
114                 label_count[label] = 0
115             label_count[label] += 1
116         ent = -sum([(p/data_length)*log(p/data_length, 2) for p in label_count.values()])
117         return ent
118
119     # 经验条件熵
120     def cond_ent(self, datasets, axis=0):
121         data_length = len(datasets)
122         feature_sets = {}
123         for i in range(data_length):
124             feature = datasets[i][axis]
125             if feature not in feature_sets:
126                 feature_sets[feature] = []
127             feature_sets[feature].append(datasets[i])
128         cond_ent = sum([(len(p)/data_length)*self.calc_ent(p) for p in feature_sets.values()])
129         return cond_ent

```

```

131     # 信息增益
132     @staticmethod
133     def info_gain(ent, cond_ent):
134         return ent - cond_ent
135
136     def info_gain_train(self, datasets):
137         count = len(datasets[0]) - 1
138         ent = self.calc_ent(datasets)
139         best_feature = []
140         for c in range(count):
141             c_info_gain = self.info_gain(ent, self.cond_ent(datasets, axis=c))
142             best_feature.append((c, c_info_gain))
143         # 比较大小
144         best_ = max(best_feature, key=lambda x: x[-1])
145         return best_
146
147     def train(self, train_data):
148         """
149         input:数据集D(DataFrame格式), 特征集A, 阈值eta
150         output:决策树T """
151         _, y_train, features = train_data.iloc[:, :-1], train_data.iloc[:, -1], train_data.columns[:-1]
152         # 1, 若D中实例属于同一类Ck, 则T为单节点树, 并将类Ck作为结点的类标记, 返回T
153         if len(y_train.value_counts()) == 1:
154             return Node(root=True,
155                         label=y_train.iloc[0])
156
157         # 2, 若A为空, 则T为单节点树, 将D中实例树最大的类Ck作为该节点的类标记, 返回T
158         if len(features) == 0:
159             return Node(root=True, label=y_train.value_counts().sort_values(ascending=False).index[0])
160
161         # 3, 计算最大信息增益 同5.1, Ag为信息增益最大的特征
162         max_feature, max_info_gain = self.info_gain_train(np.array(train_data))
163         max_feature_name = features[max_feature]
164
165         # 4, Ag的信息增益小于阈值eta, 则置T为单节点树, 并将D中是实例数最大的类Ck作为该节点的类标记, 返回T
166         if max_info_gain < self.epsilon:
167             return Node(root=True, label=y_train.value_counts().sort_values(ascending=False).index[0])
168
169         # 5, 构建Ag子集
170         node_tree = Node(root=False, feature_name=max_feature_name, feature=max_feature)
171
172         feature_list = train_data[max_feature_name].value_counts().index
173         for f in feature_list:
174             sub_train_df = train_data.loc[train_data[max_feature_name] == f].drop([max_feature_name], axis=1)
175
176             # 6, 递归生成树
177             sub_tree = self.train(sub_train_df)
178             node_tree.add_node(f, sub_tree)
179
180         # pprint.pprint(node_tree.tree)
181         return node_tree

```

```
183     def fit(self, train_data):
184         self._tree = self.train(train_data)
185         return self._tree
186
187     def predict(self, X_test):
188         return self._tree.predict(X_test)
189
190 datasets, labels = create_data()
191 data_df = pd.DataFrame(datasets, columns=labels)
192 dt = DTree()
193 tree = dt.fit(data_df)
194 print (tree)
195
196 print (dt.predict(['老年', '否', '否', '一般']))
197
```