机器学习第四章作业

任齐轩
重庆大学-计卓 2 班-20204154

日 期：September 25, 2022

1 第一题

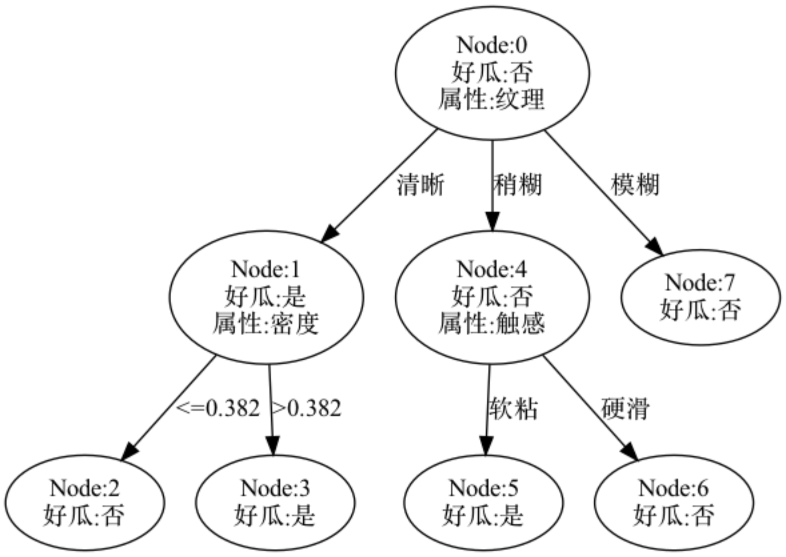
证明. 假设不存在与训练集一致的决策树，则在得到的决策树上至少存在一个节点，其上有多个不可划分的数据。而这与题意中不含冲突数据相矛盾，因此假设不成立。即必存在于训练集一致的决策树。 □

2 第二题

缺陷：可能会导致过拟合的现象，泛化性较弱

3 第三题

代码实现基于信息熵划分选择的决策树算法，得到的决策树如下图所示：代码见附录部分。



4 第七题

与图 4.2 等价的、不使用递归的决策树生成算法：

```
# 输入: 训练集 D
#       属性集 A
array[0] = [D, A]
for D, A in array:
    生成节点node;
    if D中样本全属于同一类别C:
        将node标记为C类叶节点
        continue
    elif A = 空 or D中样本在A上取值相同:
        将node标记为叶节点, 其类别标记为D中样本数最多的类
        continue
    从A中选择最优划分属性a
    for a_v in a每个取值:
        为node生成一个分支, 令D_v表示D在a上取值为a_v的样本子集
        if D_v == null:
            将分支节点标记为叶节点, 其类别标记为D中样本最多的类
            continue
        elif
            array.append([D_v, A \ {a}])
# 输出: 以node为根节点的一棵决策树
```

5 第九题

基尼值为:

$$Gini(D) = 1 - \sum_{k=1}^{|y|} \widetilde{p}_k^2 \tag{1}$$

则基尼指数为:

$$Gini\_index(D, a) = \rho * \sum_{v=1}^{|V|} \widetilde{v} Gini(\widetilde{D}^v) \tag{2}$$

A 代码

```
import numpy as np
from datasets import watermelon, watermelon_labels
from pydotplus import graphviz


class Node:
    def __init__(self, attr_init, label_init, attr_down_init):
        self.attr = attr_init
        self.label = label_init
        self.attr_down = attr_down_init


def data_load():
    data = watermelon
    labels = watermelon_labels
    return data, labels


def tot_entropy(data):
    num = len(data)
    pos_count = 0
    for i in range(num):
        if data[i][6] == '好瓜':
            pos_count += 1
    entro = -(np.log2(pos_count / num) * pos_count / num + np.log2((num -
        pos_count) / num) * (num - pos_count) / num)
    return entro


def entropy(data, label, label_index):
    num = len(data)
    pos_count = 0
    tot = 0
    for i in range(num):
        if data[i][label] == label_index:
            tot += 1
            if data[i][6] == '好瓜':
                pos_count += 1
    if pos_count / tot == 0 or pos_count / tot == 1:
        return 0
    entro = -(np.log2(pos_count / tot) * pos_count / tot + np.log2((tot -
        pos_count) / tot) * (tot - pos_count) / tot)
    return entro
```

```
def gain(data, label):
    num = len(data)
    label_box = {}
    for i in range(num):
        if data[i][label] not in label_box:
            label_box[data[i][label]] = 1
        else:
            label_box[data[i][label]] += 1
    gain_res = tot_entropy(data) - sum([label_box[i] / num * entropy(data,
        label, i) for i in label_box])

    return gain_res


def generate_tree(data, labels):
    new_node = Node(None, None, {})
    labels_count = {}
    if labels:
        for i in range(len(data)):
            if data[i][6] not in labels_count:
                labels_count[data[i][6]] = 1
            else:
                labels_count[data[i][6]] += 1
        print(labels_count)
        if len(labels_count) == 1 or len(labels_count) == 0:
            label = max(labels_count, key=labels_count.get)
            new_node.label = label
            print('label:␣', label, labels_count)
            return new_node
        index = 0
        for i in range(len(labels)):
            if gain(data, i) > gain(data, index):
                index = i
        print(index, labels)
        new_node.attr = labels[index]

        value_count = {}
        for i in range(len(data)):
            if data[i][index] not in value_count:
                value_count[data[i][index]] = 1
            else:
                value_count[data[i][index]] += 1
        for value in value_count:
            new_node.attr_down[value] = generate_tree([data[i] for i in range(
                len(data)) if data[i][index] == value],
                                                      [labels[i] for i in range
                                                          (len(labels)) if i !=
                                                              index])
    return new_node


def plot_tree(root):
    g = graphviz.Dot()
    Tree2Graph(0, g, root)
    g2 = graphviz.graph_from_dot_data(g.to_string())
    g2.write_png('tree.png')


def Tree2Graph(node_id, g, node):
    if node.label is not None:
        g.add_node(graphviz.Node(node_id, label=node.label))
        return
    g.add_node(graphviz.Node(node_id, label=node.attr))
    for value in node.attr_down:
        g.add_edge(graphviz.Edge(node_id, node_id + 1, label=value))
        Tree2Graph(node_id + 1, g, node.attr_down[value])
        node_id += 1


def main():
    data, labels = data_load()
    for i in range(len(data)):
        data[i] = [data[i][j] for j in range(len(data[i])) if j != 6 and j !=
            7]
    labels = labels[0:5]
    root = generate_tree(data, labels)
    plot_tree(root)


if __name__ == '__main__':
    main()
```