

电子科技大学
计算机科学与工程学院

标准实验报告

(实验) 课程名称 人工智能

电子科技大学

实验报告

学生姓名：朱若愚 学号：2022150501027 指导教师：段立新

实验地点：A2-413-1

实验时间：2024.5.25

一、实验室名称： 计算机学院实验中心

二、实验项目名称：决策树实验

三、实验学时：5 学时

四、实验原理：

(1) ID3 算法

ID3 算法的核心思想就是以信息增益度量属性选择，选择分裂后信息增益最大的属性进行分裂。下面先定义几个要用到的概念。设 D 为用类别对训练元组进行的划分，则 D 的熵 (entropy) 表示为：

$$\inf o(D) = - \sum_{i=1}^m p_i \log_2(p_i)$$

其中 p_i 表示第 i 个类别在整个训练元组中出现的概率，可以用属于此类别元素的数量除以训练元组元素总数量作为估计。熵的实际意义表示是 D 中元组的类标号所需要的平均信息量。现在我们假设将训练元组 D 按属性 A 进行划分，则 A 对 D 划分的期望信息为：

$$\inf o_A(D) = \sum_{j=1}^v \frac{|D_j|}{|D|} \inf o(D_j)$$

而信息增益即为两者的差值：

$$gain(A) = \inf o(D) - \inf o_A(D)$$

ID3 算法就是在每次需要分裂时，计算每个属性的增益率，然后选择增益率最大的属性进行分裂。

对于特征属性为连续值，可以如此使用 ID3 算法：先将 D 中元素按照特征属性排序，则每两个相邻元素的中间点可以看做潜在分裂点，从第一个潜在分裂点开始，分裂 D 并计算两个集合的期望信息，具有最小期望信息的点称为这个属性的最佳分裂点，其信息期望作为此属性的信息期望。

五、实验目的：

编程实现决策树算法 ID3；理解算法原理。

六、实验内容：

利用 traindata.txt 的数据（75*5，第 5 列为标签）进行训练，构造决策树；利用构造好的决策树对 testdata.txt 的数据进行分类，并输出分类准确率。

七、实验器材（设备、元器件）：

PC 微机一台

八、实验步骤：

1. 定义决策树节点 class Node:

```
6 class Node:
7     def __init__(self, feature=None, label=None):
8         self.feature = feature
9         self.label = label
10        self.children = {}
```

存储包括该节点标签以及通过字典存储子节点。

2. 读入训练数据和测试数据 load_data:

```
12 def load_data(train, test):
13     train_data = pd.read_csv(train, header=None, sep='\t')
14     test_data = pd.read_csv(test, header=None, sep='\t')
15     return train_data, test_data
16
```

使用 pandas 的 read_csv 函数读取数据，并且因为列之间以制表符隔开，用 sep='\t' 来表示。读入训练数据和测试数据储存。

3. 计算熵的函数 entropy:

```
17 def entropy(inputs):
18     total = len(inputs)
19     cou = Counter(inputs)
20     enva = 0.0
21     for label in cou:
22         pb = cou[label] / total
23         enva += pb * log2(pb)
24     return enva
```

对输入的一系列标签，先统计出总数，然后计数每一种标签的数量以及其概率，先计算出单一标签的熵进而求和计算出整个输入标签的熵。

4. 计算信息增益的函数 info_gain:

```
27 def info_gain(data, feature_idx, label_idx):
28     sorted_data = data.sort_values(by=feature_idx)
29     values = sorted_data[feature_idx].values
30     labels = sorted_data[label_idx].values
31     total_entropy = entropy(labels)
32     best_gain = 0
33     best_threshold = None
34
35     for i in range(1, len(values)):
36         if values[i] == values[i-1]:
37             continue
38         threshold = (values[i] + values[i-1]) / 2
39         left_labels = labels[:i]
40         right_labels = labels[i:]
41         left_entropy = entropy(left_labels)
42         right_entropy = entropy(right_labels)
43         weighted_entropy = (len(left_labels) / len(labels)) * left_entropy + (len(right_labels) / len(labels)) * right_entropy
44         gain = total_entropy - weighted_entropy
45         if gain > best_gain:
46             best_gain = gain
47             best_threshold = threshold
48
49     return best_gain, best_threshold
```

5. 构建决策树函数 constree:

```
51 def constree(data, label_idx, feature_indices):
52     labels = data.iloc[:, label_idx]
53     if len(set(labels)) == 1:
54         return Node(Label=labels.iloc[0])
55
56     if not feature_indices:
57         return Node(Label=labels.mode()[0])
58
59     best_feature_idx = feature_indices[0]
60     best_gain, best_threshold = info_gain(data, best_feature_idx, label_idx)
61     for feature_idx in feature_indices[1:]:
62         gain, threshold = info_gain(data, feature_idx, label_idx)
63         if gain > best_gain:
64             best_gain = gain
65             best_feature_idx = feature_idx
66             best_threshold = threshold
67
68     if best_gain == 0:
69         return Node(Label=labels.mode()[0])
70
71     tree = Node(feature=best_feature_idx, threshold=best_threshold)
72     left_data = data[data[best_feature_idx] <= best_threshold]
73     right_data = data[data[best_feature_idx] > best_threshold]
74     tree.children['<='] = constree(left_data, label_idx, feature_indices)
75     tree.children['>'] = constree(right_data, label_idx, feature_indices)
76
77     return tree
```

函数计算每个特征信息增益后选取信息增益最大的作为最佳特征，然后根据这个最佳特征创建子树。

6. 分类预测函数 classify 和 classify_data:

```
79 def classify(tree, instance):
80     if tree.Label is not None:
81         return tree.Label
82
83     feava = instance[tree.feature]
84     if feava <= tree.threshold:
85         return classify(tree.children['<='], instance)
86     else:
87         return classify(tree.children['>'], instance)
88
89 ! usage
90 def classify_data(tree, data):
91     pre = []
92     for i in range(len(data)):
93         pred = classify(tree, data.iloc[i])
94         pre.append(pred)
95
96     return pre
```

前者查找特征对应子树然后进行递归分类，后者遍历数据集使用前者进行分类，将结果添加至列表之中。

7. 结果预测准确率计算函数 accuracy:

```
96 def accuracy(pred, labels):
97     corr = 0
98     for i in range(len(pred)):
99         if pred[i] == labels.iloc[i]:
100             corr += 1
101     acc = corr / len(pred)
102     return acc
103
```

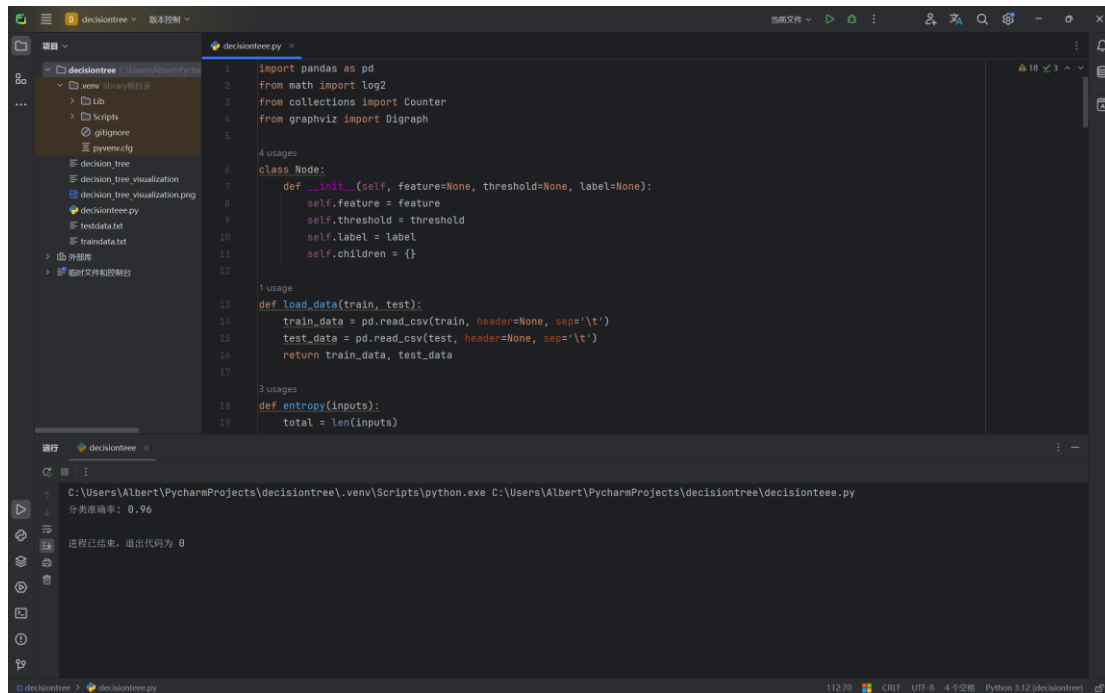
通过遍历，查找预测标签和真实标签从而计算预测准确率。

8. 可视化函数 visual:

```
104 def visual(tree, dot=None, parent=None, edge_label=None):
105     if dot is None:
106         dot = Digraph()
107         dot.attr(kw: 'graph', ranksep='0.5', nodesep='5')
108         dot.attr(kw: 'node', shape='ellipse', style='filled', fillcolor='lightblue', fontname="Helvetica", fontsize="20")
109         dot.attr(kw: 'edge', fontname="Helvetica", fontsize="30")
110
111     if tree.feature is not None:
112         dot.node(name=str(id(tree)), label=f"Feature {tree.feature}\n {tree.threshold}")
113     else:
114         dot.node(name=str(id(tree)), label=f"Label {tree.label}", shape='box', fillcolor='lightgreen')
115
116     if parent is not None:
117         dot.edge(str(parent), str(id(tree)), label=str(edge_label))
118
119     for value, child in tree.children.items():
120         visual(child, dot, id(tree), value)
121
122     return dot
```

- 1) dot.attr('graph', ranksep='0.5', nodesep='5'): 设置图形的属性。ranksep 是设置不同层之间的节点之间的距离，nodesep 是同一层的节点之间的距离。
- 2) dot.attr('node', shape='ellipse', style='filled', fillcolor='lightblue', fontname="Helvetica", fontsize="20"): 设置所有节点的默认属性。节点的形状是椭圆形，填充样式是填充，填充颜色是浅蓝色，字体是 Helvetica，字体大小是 20。
- 3) dot.attr('edge', fontname="Helvetica", fontsize="30"): 设置所有边的默认属性。边的字体是 Helvetica，字体大小是 30。
- 4) if tree.feature is not None:: 如果树的特征不为空，那么创建一个节点，标签是特征和分裂的阈值。
- 5) else:: 创建一个节点，标签，形状是矩形，颜色浅绿色。

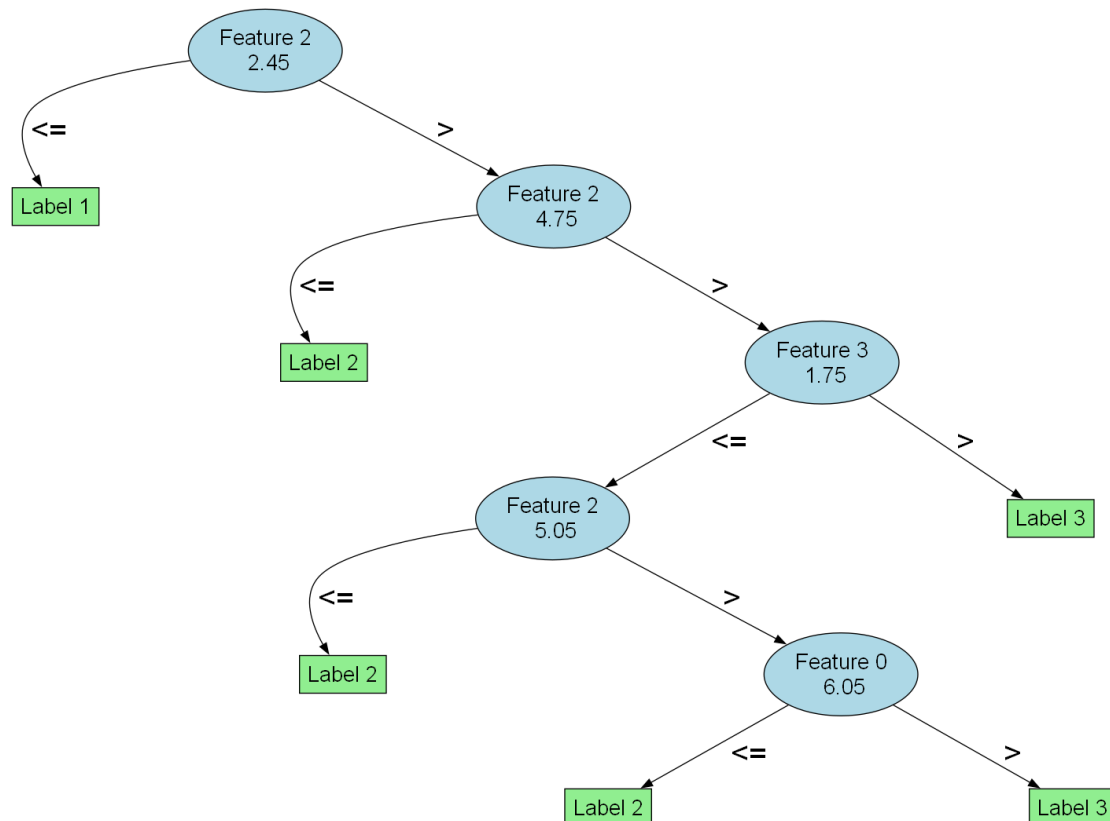
九、实验数据及结果分析：



```
1 import pandas as pd
2 from math import log2
3 from collections import Counter
4 from graphviz import Digraph
5
6 class Node:
7     def __init__(self, feature=None, threshold=None, label=None):
8         self.feature = feature
9         self.threshold = threshold
10        self.label = label
11        self.children = {}
12
13 def load_data(train, test):
14     train_data = pd.read_csv(train, header=None, sep='\t')
15     test_data = pd.read_csv(test, header=None, sep='\t')
16     return train_data, test_data
17
18 def entropy(inputs):
19     total = len(inputs)
```

分类准确率: 0.96

分类准确率: 0.96



算法通过计算信息增益，Feature2 首先分裂小于等于 2.45 即可判断为 Label 1，进一步判断大于等于 2.45 小于等于 4.75 为 Label2。若 Feature2 大于 4.75，进一步判断 Feature3，以此类推。

十、实验结论：

使用 ID3 算法可以较高准确率判断分类。

十一、总结及心得体会：

1. 决策树包含较全面，搜索较完整。
2. 健壮性较好，不受噪声影响。
3. 少量数据就可以训练出决策树。

十二、对本实验过程及方法、手段的改进建议：

通过增加训练数据集来进一步提高测试的精确度。

报告评分：

指导教师签字：