机器学习实验报告 6

姓名：陈海弘

学号：23354049

陈海弘

2024.12.21

Contents

1 摘要

本次实验主要是实现了 ID3，C4.5，CART 三种决策树算法，并且使用这三种算法对数据集进行分类。

最后做了一个决策树的实现，实现了决策树的生成和预测。

2 前置准备

本次实验包含的库有 numpy，math,pandas,matplotlib,collections,counter.

2.1 数据读取

本次实验使用的分类数据集中包含泰坦尼克号部分乘客的信息以及生还情况。数据集包括四个属性和一个标记属性，Sex，SibSp，Parch，Pclass，Survived。数据集中已经没有缺失值和异常值，所有连续变量已经离散化，标记属性 Survived 已经编码为 0 和 1。读取部分代码过于简单，不再赘述。

2.2 信息熵计算

计算信息熵公式: 某数组包含 K 个不同的取值,样本为第 k(k=1,2,...,K) 个值的数量所占比例为 p_k, 则其信息熵为

$$Ent = -\sum_{k=1}^{K} p_k log_2 p_k$$

定义一个函数 entropy，接受 label 数组，然后使用 array 将输入的数组转换为 numpy，.flatten () 将数组拉平，使用 unique 函数找到不同的取值，然后统计每个唯一值在数组中出现的次数。计算每一个 counts 的比例，最后用公式计算信息熵。

代码如下:

```
def entropy(label):
    label = np.array(label).flatten()
    _, counts = np.unique(label, return_counts = True)
    pro = counts/counts.sum()
```

```
    ent = -np.sum(pro * np.log2(pro))
    return ent
```

结果如下：

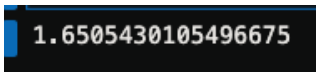

Figure 1: 信息熵计算

2.3 维度划分

为了计算信息增益，需要将数据集按照某个属性划分为不同的子集。函数将所给的数据集按照指定维度划分为若干个不同的数据集，输入属性集合、标记集合、维度索引，输出划分后得到的子数属性集合，子树标记集合。子树属性集合就是按照维度划分后的数据集，子树标记集合是对应的标记集合。

给定一个特征矩阵 `feature`、标签数组 `label`，以及划分维度 `dimension`，该函数的目标是基于指定维度的特征值将数据划分为多个子集。具体思路如下：

1. **输入处理：**
   将 `feature` 和 `label` 转换为 `NumPy` 数组，便于后续操作：

   $$\texttt{feature} \rightarrow \text{特征矩阵 } (\mathrm{m} \times \mathrm{n}), \quad \texttt{label} \rightarrow \text{标签向量 } (\mathrm{m} \times 1)$$

2. **确定唯一值：**
   提取特征矩阵第 `dimension` 列的所有唯一值，表示在该维度上划分数据的类别：

   $$\texttt{unique\_values} = \texttt{np.unique(feature[:, dimension])}$$

3. **初始化划分结果：**
   创建两个空列表，用于存储划分后的特征子集和对应的标签子集：

   $$\texttt{split\_feature} = [], \quad \texttt{split\_label} = []$$

4. **按值划分数据：**
   对于每个唯一值 value：

   (a) 构造一个布尔掩码 mask，筛选出 feature 中对应 dimension 列值等于 value 的行：

   $$\texttt{mask} = (\texttt{feature[:, dimension]} == \texttt{value})$$

   (b) 根据掩码 mask 提取符合条件的特征子集和标签子集：

   $$\texttt{split\_feature.append(feature[mask].tolist())}$$

   $$\texttt{split\_label.append(label[mask].tolist())}$$

5. **返回结果：**
   最终返回按 dimension 维度划分后的特征子集和标签子集：

   $$\texttt{return split\_feature, split\_label}$$

代码如下：

```python
def split(feature, label, dimension):
    feature = np.array(feature)
    label = np.array(label)
    unique_values = np.unique(feature[:, dimension])
    split_feature = []
    split_label = []
    for value in unique_values:
        mask = feature[:,dimension] == value
        split_feature.append(feature[mask].tolist())
        split_label.append(label[mask].tolist())

    return split_feature,split_label
```

结果如下：

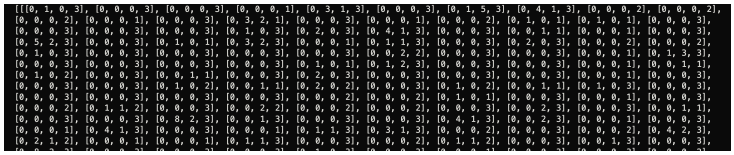

Figure 2: 属性集合

Figure 3: 标记集合

最后的属性集合和标记集合都是一个列表，列表中的元素是 numpy 数组，每个数组是一个子集。可以看到，属性集合都是依据第一个维度划分的，分别是，0，1。标记集合就是对应的标记。

3 ID3 算法

ID3 算法就是使用信息增益来选择最优的属性进行划分，信息增益的计算公式如下：

$$Gain(D, A) = Ent(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} Ent(D^v)$$

数据集 D 有属性值和标记值，样本总大小是 |D|，属性 A 有 V 个不同的取值，D^v 是 D 中 A 属性取值为 v 的样本集合，$|D^v|$ 是 D^v 的大小，Ent(D) 是数据集 D 的信息熵，Ent(D^v) 是 D^v 的信息熵。

将特征矩阵 $\mathbf{X} \in \mathbb{R}^{n \times m}$ 和标签向量 $\mathbf{y} \in \mathbb{R}^n$ 输入，目标是找到最佳的特征维度 d_{best}，以最大化信息增益 G_{best}。算法步骤如下：

1. 计算标签的总熵（Total Entropy）：

$$H_{\text{total}} = \text{Entropy}(\mathbf{y})$$

2. 初始化信息增益和分裂维度：

$$G_{\text{best}} = -\infty, \quad d_{\text{best}} = -1$$

3. 遍历每一个特征维度 $d \in [1, 2, \ldots, m]$：- 按照维度 d 对数据进行分裂，得到分裂后的特征和标签组：

$$(\mathbf{X}_d^{\text{split}}, \mathbf{y}_d^{\text{split}})$$

- 计算分裂后的条件熵（Conditional Entropy）：

$$H_{\text{conditional}} = \sum_i p_i \cdot \text{Entropy}(\mathbf{y}_i), \quad p_i = \frac{|\mathbf{y}_i|}{|\mathbf{y}|}$$

- 计算信息增益（Information Gain）：

$$G = H_{\text{total}} - H_{\text{conditional}}$$

- 如果 $G > G_{\text{best}}$，则更新：

$$G_{\text{best}} = G, \quad d_{\text{best}} = d$$

4. 返回最佳信息增益和对应的分裂维度：

$$\text{return } G_{\text{best}}, d_{\text{best}}$$

代码如下：

```
def one_split_ID3(feature, label):
    feature = np.array(feature)
    label = np.array(label)
    n_features = feature.shape[1]
    total_entropy = entropy(label)

    best_gain = -np.inf
    best_dimension = -1
```

```
9
10    for dim in range(n_features):
11        split_feature, split_label = split(feature, label, dim)
12
13        conditional_entropy = 0
14        for labels in split_label:
15            p = len(labels) / len(label)
16            conditional_entropy += p * entropy(labels)
17
18        gain = total_entropy - conditional_entropy
19
20        if gain > best_gain:
21            best_gain = gain
22            best_dimension = dim
23
24    return best_gain, best_dimension
```

4 C4.5 算法

C4.5 算法是 ID3 算法的改进版，使用信息增益率来选择最优的属性进行划分，信息增益率的计算公式如下：

$$GainRatio(D, A) = \frac{Gain(D, A)}{IV(A)}$$

将特征矩阵 $\mathbf{X} \in \mathbb{R}^{n \times m}$ 和标签向量 $\mathbf{y} \in \mathbb{R}^n$ 输入，目标是找到最佳的特征维度 d_{best}，以最大化信息增益率（Gain Ratio）。算法步骤如下：

1. 计算标签的总熵：

$$H_{\text{total}} = \text{Entropy}(\mathbf{y})$$

2. 初始化：

$$\text{最佳信息增益率} G_{\text{best}} = -\infty, \quad \text{最佳维度} d_{\text{best}} = -1$$

3. 遍历每个特征维度 $d \in [1, 2, \ldots, m]$：- 按照维度 d 对数据进行分裂，

得到分裂后的标签组 $\mathbf{y}_d^{\text{split}}$；- 计算分裂后的特征熵（Feature Entropy）：

$$H_{\text{feature}} = \sum_i \frac{|\mathbf{y}_i|}{|\mathbf{y}|} \cdot \text{Entropy}(\mathbf{y}_i)$$

其中 $\mathbf{y}_i$ 是分裂后的子集标签。

- 计算条件熵（Conditional Entropy）：

$$H_{\text{conditional}} = \sum_i \frac{|\mathbf{y}_i|}{|\mathbf{y}|} \cdot \text{Entropy}(\mathbf{y}_i)$$

- 计算信息增益（Information Gain）：

$$G = H_{\text{total}} - H_{\text{conditional}}$$

- 计算信息增益率（Gain Ratio）：

$$\text{如果} H_{\text{feature}} \neq 0, \quad \text{Gain Ratio} = \frac{G}{H_{\text{feature}}}$$

$$\text{否则，设Gain Ratio} = 0$$

- 如果当前维度的增益率大于历史最佳增益率：

$$G_{\text{best}} = \text{Gain Ratio}, \quad d_{\text{best}} = d$$

4. 返回最佳信息增益率和对应的分裂维度：

$$\text{return } G_{\text{best}}, d_{\text{best}}$$

代码展示：

```
def one_split_C4_5(feature, label):
    feature = np.array(feature)
    label = np.array(label)
    n_features = feature.shape[1]
    total_entropy = entropy(label)
```

```
6
7       best_gain_ratio = -np.inf
8       best_dimension = -1
9
10      for dim in range(n_features):
11          split_feature, split_label = split(feature, label, dim)
12
13          feature_entropy = 0
14          total_size = len(label)
15          for labels in split_label:
16              feature_entropy += (len(labels) / total_size) * entropy(
                    labels)
17
18          conditional_entropy = 0
19          for labels in split_label:
20              p = len(labels) / len(label)
21              conditional_entropy += p * entropy(labels)
22
23          gain = total_entropy - conditional_entropy
24
25          if feature_entropy != 0:
26              gain_ratio = gain / feature_entropy
27          else:
28              gain_ratio = 0
29
30          if gain_ratio > best_gain_ratio:
31              best_gain_ratio = gain_ratio
32              best_dimension = dim
33
34      return best_gain_ratio, best_dimension
```

5 CART 算法

CART 算法是一种二叉树算法，使用基尼指数来选择最优的属性进行划分，基尼指数的计算公式如下：

$$Gini(D) = 1 - \sum_{k=1}^{K} p_k^2$$

每一个属性的基尼指数是：

$$Gini(D, A) = \sum_{v=1}^{V} \frac{|D^v|}{|D|} Gini(D^v)$$

将特征矩阵 $\mathbf{X} \in \mathbb{R}^{n \times m}$ 和标签向量 $\mathbf{y} \in \mathbb{R}^n$ 输入，目标是找到最佳的特征维度 d_{best} 和分裂值 v_{best}，以最小化基尼指数（Gini Index）。算法步骤如下：

1. 初始化：

最佳基尼指数$G_{\text{best}} = \infty$, 最佳维度$d_{\text{best}} = -1$, 最佳划分值$v_{\text{best}} = \text{None}$

2. 遍历每个特征维度 $d \in [1, 2, \ldots, m]$：- 找到当前特征的所有唯一值（Unique Values）：

$$\text{unique\_values} = \text{Unique}(\mathbf{X}[:, d])$$

- 遍历每个唯一值 $v \in$ unique_values：- 按照 v 将数据分为两个子集：

左子集: $\mathbf{X}_{\text{left}}, \mathbf{y}_{\text{left}}$ 其中$\mathbf{X}[:, d] \leq v$

右子集: $\mathbf{X}_{\text{right}}, \mathbf{y}_{\text{right}}$ 其中$\mathbf{X}[:, d] > v$

- 计算两个子集的基尼系数：

$$G_{\text{left}} = 1 - \sum_{c} \left(\frac{\text{count}(c, \mathbf{y}_{\text{left}})}{|\mathbf{y}_{\text{left}}|} \right)^2$$

$$G_{\text{right}} = 1 - \sum_{c} \left(\frac{\text{count}(c, \mathbf{y}_{\text{right}})}{|\mathbf{y}_{\text{right}}|} \right)^2$$

- 计算当前划分的加权基尼指数：

$$G_{\text{index}} = \frac{|\mathbf{y}_{\text{left}}|}{|\mathbf{y}|} G_{\text{left}} + \frac{|\mathbf{y}_{\text{right}}|}{|\mathbf{y}|} G_{\text{right}}$$

- 如果 $G_{\text{index}} < G_{\text{best}}$，则更新最佳基尼指数及对应的维度和划分值：

$$G_{\text{best}} = G_{\text{index}}, \quad d_{\text{best}} = d, \quad v_{\text{best}} = v$$

3. 返回最佳基尼指数、最佳分裂维度和分裂值：

$$\text{return } G_{\text{best}}, d_{\text{best}}, v_{\text{best}}$$

代码展示：

```
1   def gini(labels):
2       total_size = len(labels)
3       if total_size == 0:
4           return 0
5       label_counts = Counter(labels)
6       gini_score = 1 - sum((count / total_size) ** 2 for count in
            label_counts.values())
7       return gini_score
8   def one_split_CART(feature, label):
9       feature = np.array(feature)
10      label = np.array(label)
11      n_features = feature.shape[1]
12
13      best_gini = np.inf
14      best_dimension = -1
15      best_value = None
16
17      for dim in range(n_features):
```

```
18          unique_values = np.unique(feature[:, dim]) # 找到每列的非重
                复值
19
20          for value in unique_values:
21              # 按当前特征值将数据划分为两个子集
22              left_indices = feature[:, dim] <= value
23              right_indices = feature[:, dim] > value
24
25              left_labels = label[left_indices]
26              right_labels = label[right_indices]
27
28              # 计算子集的基尼系数
29              left_gini = gini(left_labels)
30              right_gini = gini(right_labels)
31              total_size = len(label)
32
33              gini_index = (len(left_labels) / total_size) * left_gini
                    + \
34                          (len(right_labels) / total_size) *
                                right_gini
35
36              # 更新最小基尼系数及对应的维度和划分值
37              if gini_index < best_gini:
38                  best_gini = gini_index
39                  best_dimension = dim
40                  best_value = value
41
42      return best_gini, best_dimension, best_value
```

6 实验结果

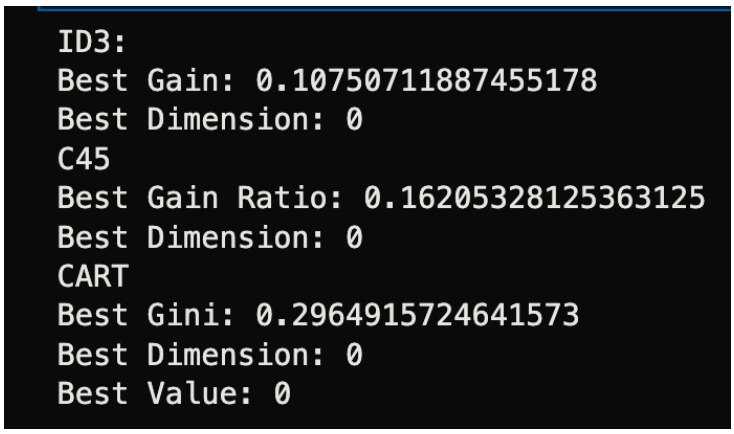

Figure 4: 结果展示

结果可以看出，这三个算法的结果算出的最佳维度都是 0，ID3 算出的最佳增益是 0.107，C4.5 算出的最佳增益率是 0.16，CART 算出的最佳基尼指数是 0.29，对应的分类值是 0。

7 决策树

这里用 ID3 算法实现决策树，完成 DTree 类中的 Tree Generate，train 函数以完成决策树的构建，并用构建好的决策树来对测试数据集进行预测并输出结果。

ID3 算法的核心思想是通过计算信息增益来选择最优属性，并根据该属性划分数据集。具体的构建过程如下：

1. **初始化：**通过传入的数据集 D，首先确定每个属性的可能取值，并初始化决策树的根节点。在代码中，这通过 'self.possible_value'字典来实现：

$$\text{self.possible\_value[attr]} = \text{np.unique}(D[:, attr])$$

这一步为每个属性记录其可能的取值。数据集 D 的最后一列为标签列，表示每个样本的类别。'D[:, -1]'表示提取标签列。

2. **计算信息熵：** 信息熵 $H(D)$ 用于衡量数据集 D 的不确定性。其公式为：

$$H(D) = -\sum_i p_i \log_2 p_i$$

其中 p_i 为类别 i 在数据集 D 中出现的概率。在代码中，信息熵通过‘entropy‘方法计算，利用 ‘np.unique‘统计不同类别的频次：

$$\text{labels, counts} = \text{np.unique}(D[:, -1], \text{return\_counts=True})$$

然后计算熵值：

$$H(D) = -\sum \left(\frac{\text{count}}{\text{total}} \times \log_2 \left(\frac{\text{count}}{\text{total}} \right) \right)$$

这里 ‘total‘是数据集 D 的样本总数。

3. **计算信息增益：** 对于每个属性，计算该属性的信息增益。信息增益 IG 表示使用该属性划分数据集后，信息熵的减少量。信息增益的公式为：

$$IG(D, A) = H(D) - H(D|A)$$

其中 $H(D|A)$ 是数据集 D 在属性 A 上的条件熵，表示在属性 A 的每个可能取值下数据集 D 的熵的加权平均。代码中，‘info_gain‘方法计算了信息增益：

$$\text{total\_entropy} = \text{self.entropy}(D)$$

然后对于每个属性的每个取值，计算条件熵并累加：

$$\text{conditional\_entropy} = \sum \left(\frac{\text{counts}[i]}{\text{total}} \times \text{self.entropy}(D[D[:, \text{attr\_index}] == \text{values}[i]]) \right)$$

最终，信息增益为总熵减去条件熵：

$$\text{info\_gain} = \text{total\_entropy} - \text{conditional\_entropy}$$

4. **选择最优属性：** 对于当前数据集 D，选择信息增益最大的属性作为划

分的标准。代码中通过以下方式选择最优属性：

$$\text{best\_attr} = \max(A, \text{key} = \lambda \text{attr} : \text{self.info\_gain}(D, \text{attr}))$$

如果信息增益为零或所有样本在某属性上的取值相同，则停止划分并将当前节点标记为叶子节点，赋值为样本类别中最多的类别。具体地，在 'TreeGenerate'方法中，若所有样本类别相同，则创建一个叶节点并返回：

$$\text{node.isLeaf} = \text{True}, \quad \text{node.label} = \text{unique\_labels}[0]$$

5. **递归构建子树：** 对于选定的最优属性，将数据集 D 按该属性的取值进行划分。然后，对于每个划分得到的子集，递归地进行相同的操作。代码中，对每个子集递归地调用 'TreeGenerate'方法：

$$\text{subset} = D[D[:, \text{best\_attr}] == \text{value}]$$

对每个子集，递归调用：

$$\text{child\_node} = \text{self.TreeGenerate}(\text{subset}, A - \{\text{best\_attr}\})$$

直到所有子集中的样本属于同一类别，或没有可用的属性进行划分。若子集为空，则创建一个叶节点，并选择出现次数最多的类别作为标签：

$$\text{child\_node} = \text{Node}(isLeaf = \text{True}, label = \text{unique\_labels}[\text{np.argmax}(counts)])$$

6. **预测：** 对于每个待预测样本，根据构建好的决策树，从根节点开始沿着属性的取值进行遍历，最终到达叶子节点，叶子节点的标签即为预测结果。代码中，通过递归遍历决策树实现预测：

$$\text{node} = \text{self.tree\_root}$$

对每个样本，沿着决策树的分支进行判断，直到到达叶节点：

$$\text{attr\_value} = \text{data}[\text{node.index}]$$

最终，通过比较预测值与真实标签，计算准确率：

$$\text{accuracy} = \frac{\text{correct}}{\text{total}}$$

代码展示：

```
class DTree:
def __init__(self):
    self.tree_root = None
    self.possible_value = {}

def entropy(self, D):
    labels, counts = np.unique(D[:, -1], return_counts=True)
    total = len(D)
    return -sum((count / total) * math.log2(count / total) for count
        in counts)

def info_gain(self, D, attr_index):
    total_entropy = self.entropy(D)
    values, counts = np.unique(D[:, attr_index], return_counts=True)
    total = len(D)
    conditional_entropy = sum((counts[i] / total) * self.entropy(D[D
        [:, attr_index] == values[i]])
                              for i in range(len(values)))
    return total_entropy - conditional_entropy

def TreeGenerate(self, D, A):
    node = Node()
    labels = D[:, -1]
    unique_labels, counts = np.unique(labels, return_counts=True)

```

```
24      if len(unique_labels) == 1:
25          node.isLeaf = True
26          node.label = unique_labels[0]
27          return node
28
29      if len(A) == 0 or all(np.all(D[:, a] == D[0, a]) for a in A):
30          node.isLeaf = True
31          node.label = unique_labels[np.argmax(counts)]
32          return node
33
34      best_attr = max(A, key=lambda attr: self.info_gain(D, attr))
35      node.isLeaf = False
36      node.index = best_attr
37
38      for value in self.possible_value[best_attr]:
39          subset = D[D[:, best_attr] == value]
40          if len(subset) == 0:
41              child_node = Node(isLeaf=True, label=unique_labels[np.
                    argmax(counts)])
42          else:
43              remaining_attrs = A - {best_attr}
44              child_node = self.TreeGenerate(subset, remaining_attrs)
45          node.addNode(value, child_node)
46
47      return node
48
49  def train(self, D):
50      D = np.array(D)
51      A = set(range(D.shape[1] - 1))
52      for attr in A:
53          self.possible_value[attr] = np.unique(D[:, attr])
54      self.tree_root = self.TreeGenerate(D, A)
55
56  def predict(self, D):
57      D = np.array(D)
```

```
    correct = 0
    total = len(D)

    for data in D:
        node = self.tree_root
        while not node.isLeaf:
            attr_value = data[node.index]
            if attr_value in node.children:
                node = node.children[attr_value]
            else:
                node = Node(isLeaf=True, label=-1)

        if node.label == data[-1]:
            correct += 1

    accuracy = correct / total
    print(f"Accuracy: {accuracy:.2f}")
```

结果展示：

Figure 5: 决策树结果

最后的结果是 0.84，说明决策树的准确率还是不错的。

8 总结

这次实验主要实现了三种决策树算法，ID3，C4.5，CART，然后实现了一个决策树的生成和预测。

ID3 算法的好处是简单，易于理解，但是缺点是容易过拟合，不能处理连续值，C4.5 算法是 ID3 的改进版，可以处理连续值，但是 C4.5 算法对于缺失值的处理不是很好，CART 算法是一种二叉树算法，可以处理连续值和缺失值，但是 CART 算法的结果是二叉树，不是多叉树。

对于数据处理的时候，需要对数据进行预处理，将连续值离散化，将缺失值填充，将标签编码，这是我容易遗忘的。