



湖南大學

HUNAN UNIVERSITY

人工智能实验六报告  
决策树

## 目录

一、 实验目的 .....	3
二、 实验描述 .....	3
三、 实验及结果分析 .....	4
(1) 开发语言及运行环境; .....	4
(2) 实验的具体步骤; .....	4
① ID3: .....	5
② C4.5: .....	6
③ CART: .....	6
(3) 根据实验数据集,按实验要求给出相应的结果(截图)并对实验结果进行简要分析。 .....	7
① ID3: .....	7
② C4.5: .....	8
③ CART: .....	9
四、 心得 .....	10
五、 程序文件名清单 .....	10
六、 附录 .....	10

## 一、实验目的

建议使用python编程实现，并在Mindspore框架下实现。

## 二、实验描述

决策树是一种监督学习算法，用于解决分类和回归问题。它通过学习数据特征和决策规则来预测未知数据的输出。

**基本思想：**

### ID3算法 (Iterative Dichotomiser 3)

1. 核心思想：ID3算法使用信息增益 (Information Gain) 作为标准来构建决策树。信息增益衡量了特征对于分类结果的不确定性减少的程度。
2. 信息增益计算：对于一个数据集，计算每个特征的信息增益，选择信息增益最大的特征作为节点分裂的依据。
3. 停止条件：所有实例属于同一类别，或者数据集中没有更多的特征可以用于进一步分裂。
4. 特点：ID3算法只能处理分类问题，不能处理连续值特征，且容易过拟合。

### C4.5算法

1. 起源：C4.5是ID3算法的改进版，由Ross Quinlan在1993年提出。
2. 核心思想：C4.5算法同样使用信息增益作为构建决策树的标准，但引入了增益率 (Gain Ratio) 来克服ID3对可取值数目较多的特征的偏好。
3. 增益率计算：增益率是信息增益与分裂信息 (分裂信息衡量了特征分裂后数据集的不确定性) 的比值。
4. 处理连续值：C4.5可以处理连续值特征，通过二分法来选择最佳分裂点。
5. 处理缺失值：C4.5可以处理数据集中的缺失值。
6. 输出：C4.5生成的决策树是多路树 (每个节点可以有多个子节点)，而ID3生成的是二叉树。
7. 特点：C4.5可以处理分类和回归问题，但仍然容易过拟合。

### CART算法 (Classification and Regression Trees)

1. 起源：CART算法由Breiman等人在1984年提出，是一种二元分裂的决策树算法。
2. 核心思想：CART算法既可以用于分类问题，也可以用于回归问题。对于分

类问题，CART使用基尼不纯度（Gini Impurity）作为分裂标准；对于回归问题，使用平方误差和（Variance Reduction）。

3. 基尼不纯度计算：基尼不纯度衡量了数据集的不纯度，值越小表示数据集的纯度越高。

4. 处理连续值：CART可以处理连续值特征，通过选择最佳分裂点来分裂节点。

5. 输出：CART生成的是二叉树，每个节点都有两个子节点。

6. 剪枝：CART在构建树之后通常会进行剪枝操作，以防止过拟合。

7. 特点：CART算法可以处理分类和回归问题，且通过剪枝操作减少了过拟合的风险。

### **数据集处理：**

main中，使用pd.read\_csv读取数据，并进行数据预处理，然后再进行训练。

## **三、实验及结果分析**

### **（1）开发语言及运行环境；**

与实验一相同，不再赘述。

### **（2）实验的具体步骤；**

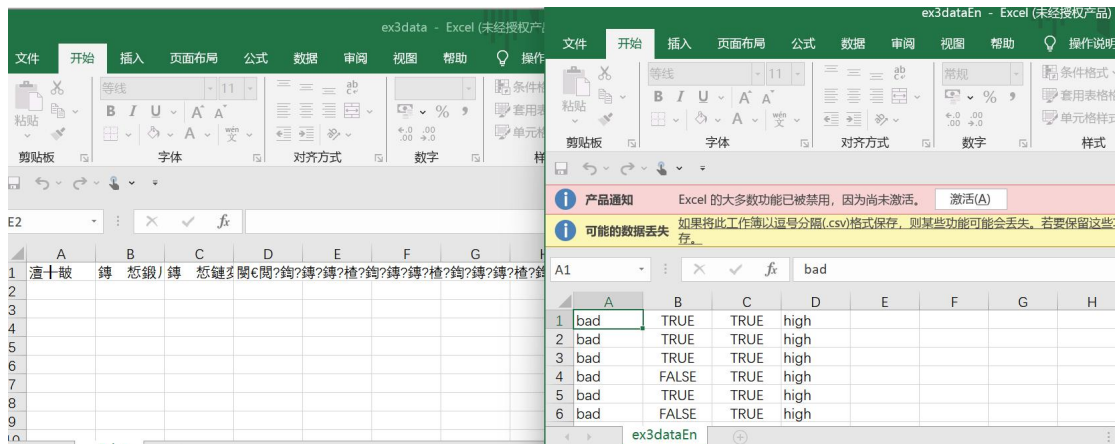
#### **题目：**

某连锁餐饮企业想了解周末和非周末对销量是否有很大影响，以及天气好坏、是否有促销活动对销量的影响，从而公司的辅助决策。现有单个门店的历史数据。请按要求完成实验。

#### **数据集：**

文件ex3data.xls 为该实验的数据集，第1-5 列分别表示序号、天气好坏、是否周末、是否有促销和销量高低。

它的中文版好像有问题，所以使用英文版的数据集ex3dataEn.xls。而且内容上只有四列，序号使用的是表格自带的。



## 步骤与要求:

选择ID3、C4.5、CART 三种常见决策树算法中的一种建立决策树模型，并画出决策树。（采用多种算法实现并进行算法的比较分析者，将获得更高分数）

### ① ID3:

OptimizedID3DecisionTree 类是 ID3 决策树算法的实现。

```

12  class OptimizedID3DecisionTree:
13      def __init__(self):
14          self.tree = None
15          self.level = 0 # 用于缩进显示
16
17      2 usages
18      def calc_entropy(self, y):
19          unique_vals = np.unique(y)

```

self.tree 和 self.level 用于打印树结构。calc\_info\_gain 用于计算信息增益，并打印出每个特征值对应的熵和权重，以及最终的信息增益值。

熵的计算：公式-hlogh

```

17      def calc_entropy(self, y):
18          unique_vals = np.unique(y)
19          entropy = 0
20          for val in unique_vals:
21              p = len(y[y == val]) / len(y)
22              entropy -= p * np.log2(p)
23          return entropy

```

\_build\_tree 是最关键的部分，它构建决策树，调用函数进行信息增益的计算和最佳特征的选择。

检查终止条件：如果样本都属于同一类别或没有更多特征可用，则创建叶节点。否则，方法计算每个特征的信息增益，选择信息增益最大的特征作为分裂节

点，并打印相关信息。

对于所选特征的每个值，它创建子节点并递归地对每个子集应用相同的分裂过程，直到所有样本都不能再分裂。递归过程中，`self.level` 控制打印语句的缩进，以显示树的层次结构。如果遇到未知特征值，方法会返回多数类预测结果。

`predict` 和 `_predict_single` 方法进行预测。

最后 `plot_optimized_tree` 绘制决策树。

## ② C4.5:

`OptimizedC45DecisionTree` 类是 C4.5 决策树算法的实现，他是 ID3 的改进版本。它和 ID3 的区别是，它使用增益率而不是信息增益来选择分裂特征，这使得算法对有更多值的特征不那么敏感。

```
12  class OptimizedC45DecisionTree:
13      def __init__(self):
14          self.tree = None
15          self.level = 0 # 用于缩进显示
16
17      def calc_entropy(self, y):
```

C4.5 的算法和 ID3 很像，只有部分内容不同，比如 `_build_tree` 的时候使用的是信息增益率。

信息增益率的计算：信息增益与分裂信息的比值。`calc_entropy` 计算信息增益，函数 `calc_split_info` 计算分裂信息。

```
def calc_gain_ratio(self, X, y, feature, print_details=True):
    """计算增益率"""
    info_gain = self.calc_info_gain(X, y, feature, print_details)
    split_info = self.calc_split_info(X, feature, print_details)

    # 避免除以零
    if split_info == 0:
        gain_ratio = 0
    else:
        gain_ratio = info_gain / split_info

    if print_details:
        indent = " " * self.level
        print(f"{indent}增益率 = {gain_ratio:.4f}")

    return gain_ratio
```

## ③ CART:

`OptimizedCARTDecisionTree` 类是 CART 算法的实现，该算法通过递归地划分数据集来构建树结构。

```

9  class OptimizedCARTDecisionTree:
10     def __init__(self, max_depth=5, min_samples_split=2):
11         self.tree = None
12         self.max_depth = max_depth
13         self.min_samples_split = min_samples_split

```

`self.max_depth` 控制树的最大深度, `self.min_samples_split` 定义了分裂一个节点所需的最小样本数。

`calc_gini` 计算给定数据集的基尼指数, 基尼指数是衡量数据集不纯度的一种方法, 值越小表示数据集的纯度越高。

```

15  def calc_gini(self, y):
16      """ 计算基尼指数 """
17      if len(y) == 0:
18          return 0
19      _, counts = np.unique(y, return_counts=True)
20      probabilities = counts / len(y)
21      return 1 - np.sum(probabilities ** 2)

```

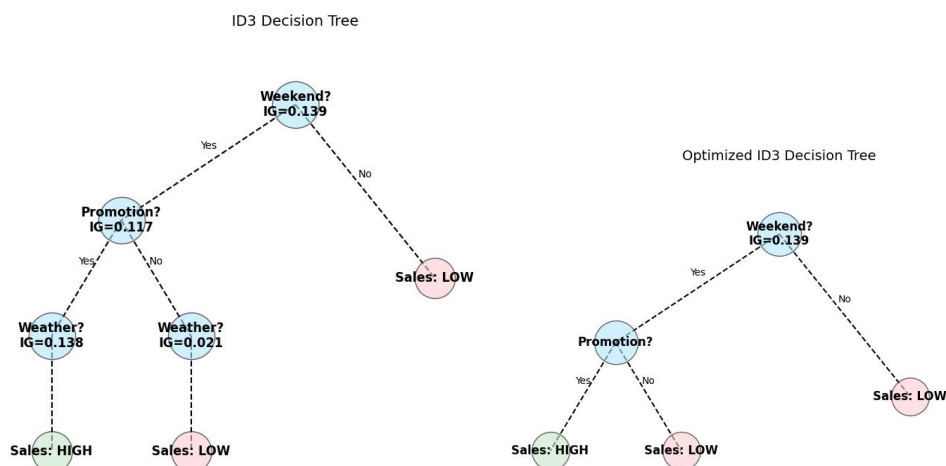
`find_best_split` 寻找给定数据集的最佳分裂特征和分裂点。它遍历所有特征和可能的分裂点, 计算每个分裂的基尼增益, 并选择基尼增益最大的分裂。

`_build_tree`中, 创建决策树之后, 会对左右子节点进行检查, 若子节点类别相同, 则合并, 可以让树好看很多。

其他基本相同。

**(3) 根据实验数据集, 按实验要求给出相应的结果(截图)并对实验结果进行简要分析。**

### ① ID3:



优化前，由于是递归生成的，所以到 **weather** 这一层时，即便是分类结果相同，也没有合并，直接显示（正常情况下如果分类结果不同还有分支是会对 **weather** 进行 **yes** 和 **no** 的判断的）；优化后添加了是否可以合并节点的检查，优化了决策树。

**ID3决策树准确率：0.7353**

ID3 的准确率有 0.7353.

开始构建决策树...

计算每个特征的信息增益：

计算特征 '**weather**' 的信息增益：

父节点熵 = 0.9975

当**weather**=坏：

样本数：17

熵：0.9774

权重：0.5000

当**weather**=好：

样本数：17

熵：0.9367

权重：0.5000

信息增益 = 0.0405

选择信息增益最大的特征：**weekend** (IG = 0.1394)

创建以 **weekend** 为分裂特征的节点

处理 **weekend**=否 的分支：

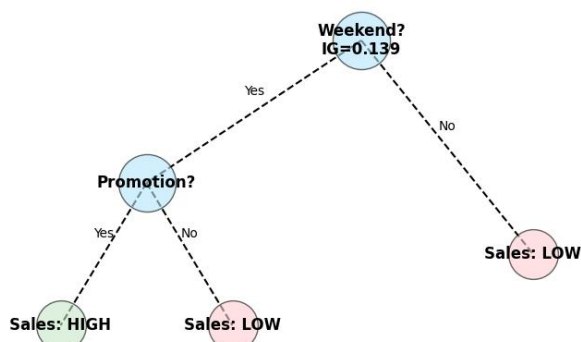
**选择信息增益最大的特征：weather (IG = 0.1379)**

**所有分支结果相同（销量高），合并为叶节点**

计算时会对当前特征的信息增益进行输出，并递归对子节点进行检查，当分支计算完毕后会检测结果进行检测，若结果相同将合并为叶节点。

## ② C4.5:

Optimized ID3 Decision Tree





可以看到这里 C4.5 和 ID3 给出的决策树结果基本相同。

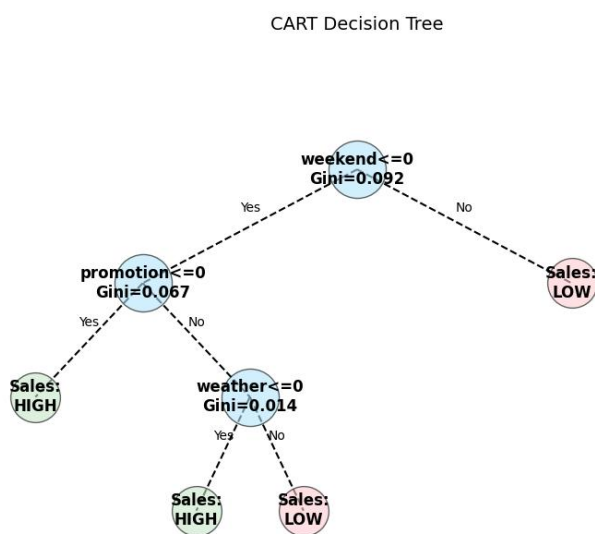
```
计算特征 'weather' 的信息增益:  
父节点熵 = 0.9975  
当weather=坏:  
  样本数: 17  
  熵: 0.9774  
  权重: 0.5000  
当weather=好:  
  样本数: 17  
  熵: 0.9367  
  权重: 0.5000  
信息增益 = 0.0405  
计算分裂信息:  
当weather=坏:  
  样本数: 17  
  比例: 0.5000  
当weather=好:  
  样本数: 17  
  比例: 0.5000  
分裂信息 = 1.0000  
增益率 = 0.0405
```

相比 ID3, C4.5 还会输出分裂信息的计算, 其他基本相同, 都是选择增益率最大的特征, 并且会对分支结果进行合并。

**C4.5决策树准确率: 0.7353**

它的准确率也相同。

### ③ CART:



CART 给出的决策树更详细一点, 对于 promotion 的 no 结果, 还详细分了 weather 的情况。

**CART决策树准确率：0.7353**

CART 的准确率也是 0.7353，本来以为会更高一点的，但是计算准确率调用的是库函数 `accuracy_score`，应该不会有问题。

ID3 算法使用信息增益作为分裂标准，倾向于选择具有更多可能值的特征进行分裂，而 C4.5 算法通过增益率来减少对具有更多可能值的特征的偏好，更适合处理具有不同值数量的特征。CART 算法则使用基尼指数作为分裂标准，能够自然地处理连续特征，并通过选择最佳分裂点来分裂连续特征，同时 CART 通常使用后剪枝技术来控制过拟合。在决策树结构上，三种算法都以 "Weekend?" 作为根节点，但度量标准不同，ID3 展示信息增益，C4.5 展示增益率，CART 展示基尼指数

## 四、心得

实验做到一半 300 代金券用完了，天都塌了。半夜短信提醒我已经欠费了，而且：

### 4. 代金券不能用来核销欠费。

被迫充了 11 块钱。

去华为云沃土云创计划又领了学生权益 401 块的代金券，还好能用。

## 五、程序文件名清单

`mainID3.py`: ID3 源代码

`mainID3.py.pdf`: ID3 源代码 pdf 版

`mainC45.py`: C45 源代码

`mainC45.py.pdf`: C45 源代码 pdf 版

`mainCART.py`: CART 源代码

`mainCART.py.pdf`: CART 源代码 pdf 版

## 六、附录

代码如下。

## mainID3.py

```
1  import numpy as np
2  import pandas as pd
3  from mindspore import context
4  import matplotlib.pyplot as plt
5  from sklearn.preprocessing import LabelEncoder
6  from sklearn.metrics import accuracy_score
7
8  # 设置MindSpore上下文
9  context.set_context(mode=context.PYNATIVE_MODE, device_target="CPU")
10
11
12  class OptimizedID3DecisionTree:
13      def __init__(self):
14          self.tree = None
15          self.level = 0 # 用于缩进显示
16
17      def calc_entropy(self, y):
18          unique_vals = np.unique(y)
19          entropy = 0
20          for val in unique_vals:
21              p = len(y[y == val]) / len(y)
22              entropy -= p * np.log2(p)
23          return entropy
24
25      def calc_info_gain(self, X, y, feature, print_details=True):
26          base_entropy = self.calc_entropy(y)
27          unique_vals = np.unique(X[feature])
28          weighted_entropy = 0
29
30          if print_details:
31              indent = " " * self.level
32              print(f"\n{indent}计算特征 '{feature}' 的信息增益:")
33              print(f"{indent}父节点熵 = {base_entropy:.4f}")
34
35          for val in unique_vals:
36              subset_idx = X[feature] == val
37              subset_y = y[subset_idx]
38              subset_entropy = self.calc_entropy(subset_y)
39              weight = len(subset_y) / len(y)
40              weighted_entropy += weight * subset_entropy
41
42          if print_details:
43              val_name = "是" if val == 1 else "否" if val == 0 else val
44              if feature == 'weather':
45                  val_name = "好" if val == 1 else "坏"
46              print(f"{indent} 当 {feature}={val_name}:")
47              print(f"{indent} 样本数: {len(subset_y)}")
48              print(f"{indent} 熵: {subset_entropy:.4f}")
49              print(f"{indent} 权重: {weight:.4f}")
50
51          info_gain = base_entropy - weighted_entropy
52          if print_details:
53              print(f"{indent}信息增益 = {info_gain:.4f}")
54
55          return info_gain
56
57      def _get_subset_predictions(self, X, y, feature):
58          unique_vals = np.unique(X[feature])
59          predictions = {}
60
61          for val in unique_vals:
62              subset_idx = X[feature] == val
63              if len(y[subset_idx]) > 0:
64                  if len(np.unique(y[subset_idx])) == 1:
65                      predictions[val] = y[subset_idx].iloc[0]
66                  else:
67                      predictions[val] = y[subset_idx].mode()[0]
68
69          return predictions
70
71      def _should_merge_nodes(self, predictions):
```

```

72         return len(set(predictions.values())) == 1
73
74     def fit(self, X, y):
75         print("\n开始构建决策树...")
76         self.tree = self._build_tree(X, y)
77         return self
78
79     def _build_tree(self, X, y):
80         indent = " " * self.level
81
82         if len(np.unique(y)) == 1:
83             result = "高" if y.iloc[0] == 1 else "低"
84             print(f"{indent}所有样本销量均为{result}，创建叶节点")
85             return {'prediction': y.iloc[0]}
86
87         if X.empty:
88             result = "高" if y.mode()[0] == 1 else "低"
89             print(f"{indent}没有更多特征，使用多数类（销量{result}）创建叶节点")
90             return {'prediction': y.mode()[0]}
91
92         info_gains = {}
93         predictions_by_feature = {}
94
95         print(f"\n{indent}计算每个特征的信息增益:")
96         for feature in X.columns:
97             info_gains[feature] = self.calc_info_gain(X, y, feature)
98             predictions_by_feature[feature] = self._get_subset_predictions(X, y, feature)
99
100        best_feature = max(info_gains, key=info_gains.get)
101        print(f"\n{indent}选择信息增益最大的特征: {best_feature} (IG = {info_gains[best_feature]:.4f})")
102
103        if self._should_merge_nodes(predictions_by_feature[best_feature]):
104            pred_val = next(iter(predictions_by_feature[best_feature].values()))
105            result = "高" if pred_val == 1 else "低"
106            print(f"{indent}所有分支结果相同（销量{result}），合并为叶节点")
107            return {'prediction': pred_val}
108
109        tree = {'feature': best_feature, 'info_gain': info_gains[best_feature]}
110
111        print(f"{indent}创建以 {best_feature} 为分裂特征的节点")
112        self.level += 1
113
114        for value in np.unique(X[best_feature]):
115            subset_idx = X[best_feature] == value
116            val_name = "是" if value == 1 else "否" if value == 0 else value
117            if best_feature == 'weather':
118                val_name = "好" if value == 1 else "坏"
119            print(f"\n{indent}处理 {best_feature}={val_name} 的分支:")
120
121            if len(y[subset_idx]) == 0:
122                result = "高" if y.mode()[0] == 1 else "低"
123                print(f"{indent} 没有对应样本，使用多数类（销量{result}）创建叶节点")
124                tree[value] = {'prediction': y.mode()[0]}
125            else:
126                X_subset = X[subset_idx].drop(columns=[best_feature])
127                tree[value] = self._build_tree(X_subset, y[subset_idx])
128
129        self.level -= 1
130        return tree
131
132    def predict(self, X):
133        """进行预测"""
134        y_pred = []
135        for _, row in X.iterrows():
136            pred = self._predict_single(row, self.tree)
137            y_pred.append(pred)
138        return np.array(y_pred)
139
140    def _predict_single(self, x, tree):
141        """对单个样本进行预测"""
142        # 如果到达叶节点，返回预测值
143        if 'prediction' in tree:
144            return tree['prediction']
145

```



```
146     # 获取当前节点的特征
147     feature = tree['feature']
148     value = x[feature]
149
150     # 如果特征值存在于树中，继续向下遍历
151     if value in tree:
152         return self._predict_single(x, tree[value])
153     else:
154         # 如果遇到未知的特征值，返回这个节点下所有叶节点的多数类
155         leaf_values = []
156         for subtree in tree.values():
157             if isinstance(subtree, dict) and 'prediction' in subtree:
158                 leaf_values.append(subtree['prediction'])
159
160         if leaf_values:
161             return max(set(leaf_values), key=leaf_values.count)
162     return 1 # 默认返回高销量类别
163
164
165 def plot_optimized_tree(tree, ax):
166     """绘制决策树"""
167     decision_node_color = '#B3E5FC' # 浅蓝色决策节点
168     leaf_node_high_color = '#C8E6C9' # 浅绿色高销量节点
169     leaf_node_low_color = '#FFCDD2' # 浅红色低销量节点
170
171     def draw_node(x, y, content, node_type='decision'):
172         if node_type == 'decision':
173             color = decision_node_color
174             size = 2000
175         else:
176             color = leaf_node_high_color if node_type == 'high' else leaf_node_low_color
177             size = 1500
178
179         ax.scatter(x, y, s=size, c=color, edgecolor='black', alpha=0.6, zorder=2)
180         ax.text(x, y, content, ha='center', va='center', color='black',
181                fontsize=12, fontweight='bold')
182
183     def draw_connection(start, end, text=''):
184         ax.plot([start[0], end[0]], [start[1], end[1]], 'black',
185                linestyle='--', zorder=1)
186         if text:
187             mid_x = (start[0] + end[0]) / 2
188             mid_y = (start[1] + end[1]) / 2
189             ax.text(mid_x, mid_y + 0.1, text, ha='center', va='bottom',
190                    color='black', fontsize=10)
191
192     # 设置图形属性
193     ax.set_title('Optimized ID3 Decision Tree', pad=20, size=14)
194     ax.set_xlim(-5, 5)
195     ax.set_ylim(0, 4)
196     ax.axis('off')
197
198     # 绘制树结构
199     # 根节点
200     draw_node(0, 3.5, f"Weekend?\nIG={tree['info_gain']:.3f}")
201
202     # 第二层节点
203     draw_node(-2.5, 2.5, "Promotion?") # Weekend = Yes
204     draw_node(2.0, 2.0, "Sales: LOW", 'low') # Weekend = No
205
206     # 第三层节点
207     draw_node(-3.5, 1.5, "Sales: HIGH", 'high') # Weekend = Yes, Promotion = Yes
208     draw_node(-1.5, 1.5, "Sales: LOW", 'low') # Weekend = Yes, Promotion = No
209
210     # 绘制连接线
211     draw_connection((0, 3.5), (-2.5, 2.5), 'Yes')
212     draw_connection((0, 3.5), (2.0, 2.0), 'No')
213     draw_connection((-2.5, 2.5), (-3.5, 1.5), 'Yes')
214     draw_connection((-2.5, 2.5), (-1.5, 1.5), 'No')
215
216
217 def main():
218     # 读取数据
219     data = pd.read_csv('ex3dataEn.csv', header=None,
```

```
220         names=['weather', 'weekend', 'promotion', 'sales'])
221
222     # 数据预处理
223     le = LabelEncoder()
224     for column in data.columns:
225         data[column] = le.fit_transform(data[column])
226
227     X = data.iloc[:, :3]
228     y = data.iloc[:, 3]
229
230     # 训练优化后的ID3决策树
231     id3_tree = OptimizedID3DecisionTree()
232     id3_tree.fit(X, y)
233
234     # 预测和计算准确率
235     y_pred = id3_tree.predict(X)
236     accuracy = accuracy_score(y, y_pred)
237
238     # 创建决策树可视化
239     fig, ax = plt.subplots(figsize=(12, 8))
240     plot_optimized_tree(id3_tree.tree, ax)
241
242     plt.savefig('optimized_id3_decision_tree.png', dpi=300, bbox_inches='tight',
243               facecolor='white', edgecolor='none')
244     plt.show()
245
246     # 输出准确率
247     print(f"\nID3决策树准确率: {accuracy:.4f}")
248
249
250 if __name__ == "__main__":
251     main()
```

## mainC45.py

```
1  import numpy as np
2  import pandas as pd
3  from mindspore import context
4  import matplotlib.pyplot as plt
5  from sklearn.preprocessing import LabelEncoder
6  from sklearn.metrics import accuracy_score
7
8  # 设置MindSpore上下文
9  context.set_context(mode=context.PYNATIVE_MODE, device_target="CPU")
10
11
12  class OptimizedC45DecisionTree:
13      def __init__(self):
14          self.tree = None
15          self.level = 0 # 用于缩进显示
16
17      def calc_entropy(self, y):
18          unique_vals = np.unique(y)
19          entropy = 0
20          for val in unique_vals:
21              p = len(y[y == val]) / len(y)
22              entropy -= p * np.log2(p)
23          return entropy
24
25      def calc_split_info(self, X, feature, print_details=True):
26          """计算分裂信息"""
27          unique_vals = np.unique(X[feature])
28          split_info = 0
29          total_samples = len(X)
30
31          if print_details:
32              indent = " " * self.level
33              print(f"{indent}计算分裂信息:")
34
35          for val in unique_vals:
36              subset_size = len(X[X[feature] == val])
37              p = subset_size / total_samples
38              split_info -= p * np.log2(p)
39
40          if print_details:
41              val_name = "是" if val == 1 else "否" if val == 0 else val
42              if feature == 'weather':
43                  val_name = "好" if val == 1 else "坏"
44              print(f"{indent} 当 {feature}={val_name}:")
45              print(f"{indent}    样本数: {subset_size}")
46              print(f"{indent}    比例: {p:.4f}")
47
48          if print_details:
49              print(f"{indent}分裂信息 = {split_info:.4f}")
50
51          return split_info
52
53      def calc_info_gain(self, X, y, feature, print_details=True):
54          base_entropy = self.calc_entropy(y)
55          unique_vals = np.unique(X[feature])
56          weighted_entropy = 0
57
58          if print_details:
59              indent = " " * self.level
60              print(f"\n{indent}计算特征 '{feature}' 的信息增益:")
61              print(f"{indent}父节点熵 = {base_entropy:.4f}")
62
63          for val in unique_vals:
64              subset_idx = X[feature] == val
65              subset_y = y[subset_idx]
66              subset_entropy = self.calc_entropy(subset_y)
67              weight = len(subset_y) / len(y)
68              weighted_entropy += weight * subset_entropy
69
70          if print_details:
71              val_name = "是" if val == 1 else "否" if val == 0 else val
72              if feature == 'weather':
73                  val_name = "好" if val == 1 else "坏"
74              print(f"{indent} 当 {feature}={val_name}:")
75              print(f"{indent}    样本数: {len(subset_y)}")
```



```

76         print(f"{indent}    熵: {subset_entropy:.4f}")
77         print(f"{indent}    权重: {weight:.4f}")
78
79     info_gain = base_entropy - weighted_entropy
80     if print_details:
81         print(f"{indent}信息增益 = {info_gain:.4f}")
82
83     return info_gain
84
85 def calc_gain_ratio(self, X, y, feature, print_details=True):
86     """计算增益率"""
87     info_gain = self.calc_info_gain(X, y, feature, print_details)
88     split_info = self.calc_split_info(X, feature, print_details)
89
90     # 避免除以零
91     if split_info == 0:
92         gain_ratio = 0
93     else:
94         gain_ratio = info_gain / split_info
95
96     if print_details:
97         indent = " " * self.level
98         print(f"{indent}增益率 = {gain_ratio:.4f}")
99
100    return gain_ratio
101
102 def _get_subset_predictions(self, X, y, feature):
103     unique_vals = np.unique(X[feature])
104     predictions = {}
105
106     for val in unique_vals:
107         subset_idx = X[feature] == val
108         if len(y[subset_idx]) > 0:
109             if len(np.unique(y[subset_idx])) == 1:
110                 predictions[val] = y[subset_idx].iloc[0]
111             else:
112                 predictions[val] = y[subset_idx].mode()[0]
113
114     return predictions
115
116 def _should_merge_nodes(self, predictions):
117     return len(set(predictions.values())) == 1
118
119 def fit(self, X, y):
120     print("\n开始构建C4.5决策树...")
121     self.tree = self._build_tree(X, y)
122     return self
123
124 def _build_tree(self, X, y):
125     indent = " " * self.level
126
127     if len(np.unique(y)) == 1:
128         result = "高" if y.iloc[0] == 1 else "低"
129         print(f"{indent}所有样本销量均为{result}，创建叶节点")
130         return {'prediction': y.iloc[0]}
131
132     if X.empty:
133         result = "高" if y.mode()[0] == 1 else "低"
134         print(f"{indent}没有更多特征，使用多数类（销量{result}）创建叶节点")
135         return {'prediction': y.mode()[0]}
136
137     gain_ratios = {}
138     predictions_by_feature = {}
139
140     print(f"\n{indent}计算每个特征的增益率:")
141     for feature in X.columns:
142         gain_ratios[feature] = self.calc_gain_ratio(X, y, feature)
143         predictions_by_feature[feature] = self._get_subset_predictions(X, y, feature)
144
145     best_feature = max(gain_ratios, key=gain_ratios.get)
146     print(f"\n{indent}选择增益率最大的特征: {best_feature} (GainRatio = {gain_ratios[best_feature]:.4f})")
147
148     if self._should_merge_nodes(predictions_by_feature[best_feature]):
149         pred_val = next(iter(predictions_by_feature[best_feature].values()))
150         result = "高" if pred_val == 1 else "低"
151         print(f"{indent}所有分支结果相同（销量{result}），合并为叶节点")
152         return {'prediction': pred_val}
153

```



```

154     tree = {'feature': best_feature, 'gain_ratio': gain_ratios[best_feature]}
155
156     print(f"{indent}创建以 {best_feature} 为分裂特征的节点")
157     self.level += 1
158
159     for value in np.unique(X[best_feature]):
160         subset_idx = X[best_feature] == value
161         val_name = "是" if value == 1 else "否" if value == 0 else val
162         if best_feature == 'weather':
163             val_name = "好" if value == 1 else "坏"
164         print(f"\n{indent}处理 {best_feature}={val_name} 的分支:")
165
166         if len(y[subset_idx]) == 0:
167             result = "高" if y.mode()[0] == 1 else "低"
168             print(f"{indent} 没有对应样本, 使用多数类 (销量 {result}) 创建叶节点")
169             tree[value] = {'prediction': y.mode()[0]}
170         else:
171             X_subset = X[subset_idx].drop(columns=[best_feature])
172             tree[value] = self._build_tree(X_subset, y[subset_idx])
173
174     self.level -= 1
175     return tree
176
177 def predict(self, X):
178     """进行预测"""
179     y_pred = []
180     for _, row in X.iterrows():
181         pred = self._predict_single(row, self.tree)
182         y_pred.append(pred)
183     return np.array(y_pred)
184
185 def _predict_single(self, x, tree):
186     """对单个样本进行预测"""
187     if 'prediction' in tree:
188         return tree['prediction']
189
190     feature = tree['feature']
191     value = x[feature]
192
193     if value in tree:
194         return self._predict_single(x, tree[value])
195     else:
196         leaf_values = []
197         for subtree in tree.values():
198             if isinstance(subtree, dict) and 'prediction' in subtree:
199                 leaf_values.append(subtree['prediction'])
200
201         if leaf_values:
202             return max(set(leaf_values), key=leaf_values.count)
203         return 1 # 默认返回高销量类别
204
205
206 def plot_optimized_tree(tree, ax):
207     """绘制决策树"""
208     decision_node_color = '#B3E5FC' # 浅蓝色决策节点
209     leaf_node_high_color = '#C8E6C9' # 浅绿色高销量节点
210     leaf_node_low_color = '#FFCDD2' # 浅红色低销量节点
211
212     def draw_node(x, y, content, node_type='decision'):
213         if node_type == 'decision':
214             color = decision_node_color
215             size = 2000
216         else:
217             color = leaf_node_high_color if node_type == 'high' else leaf_node_low_color
218             size = 1500
219
220         ax.scatter(x, y, s=size, c=color, edgecolor='black', alpha=0.6, zorder=2)
221         ax.text(x, y, content, ha='center', va='center', color='black',
222                fontsize=12, fontweight='bold')
223
224     def draw_connection(start, end, text=''):
225         ax.plot([start[0], end[0]], [start[1], end[1]], 'black',
226                linestyle='--', zorder=1)
227         if text:
228             mid_x = (start[0] + end[0]) / 2
229             mid_y = (start[1] + end[1]) / 2
230             ax.text(mid_x, mid_y + 0.1, text, ha='center', va='bottom',
231                    color='black', fontsize=10)

```

```
232
233     # 设置图形属性
234     ax.set_title('Optimized C4.5 Decision Tree', pad=20, size=14)
235     ax.set_xlim(-5, 5)
236     ax.set_ylim(0, 4)
237     ax.axis('off')
238
239     # 绘制树结构
240     # 根节点
241     draw_node(0, 3.5, f"Weekend?\nGR={tree['gain_ratio']:.3f}")
242
243     # 第二层节点
244     draw_node(-2.5, 2.5, "Promotion?") # Weekend = Yes
245     draw_node(2.0, 2.0, "Sales: LOW", 'low') # Weekend = No
246
247     # 第三层节点
248     draw_node(-3.5, 1.5, "Sales: HIGH", 'high') # Weekend = Yes, Promotion = Yes
249     draw_node(-1.5, 1.5, "Sales: LOW", 'low') # Weekend = Yes, Promotion = No
250
251     # 绘制连接线
252     draw_connection((0, 3.5), (-2.5, 2.5), 'Yes')
253     draw_connection((0, 3.5), (2.0, 2.0), 'No')
254     draw_connection((-2.5, 2.5), (-3.5, 1.5), 'Yes')
255     draw_connection((-2.5, 2.5), (-1.5, 1.5), 'No')
256
257
258 def main():
259     # 读取数据
260     data = pd.read_csv('ex3dataEn.csv', header=None,
261                       names=['weather', 'weekend', 'promotion', 'sales'])
262
263     # 数据预处理
264     le = LabelEncoder()
265     for column in data.columns:
266         data[column] = le.fit_transform(data[column])
267
268     X = data.iloc[:, :3]
269     y = data.iloc[:, 3]
270
271     # 训练C4.5决策树
272     c45_tree = OptimizedC45DecisionTree()
273     c45_tree.fit(X, y)
274
275     # 预测和计算准确率
276     y_pred = c45_tree.predict(X)
277     accuracy = accuracy_score(y, y_pred)
278
279     # 创建决策树可视化
280     fig, ax = plt.subplots(figsize=(12, 8))
281     plot_optimized_tree(c45_tree.tree, ax)
282
283     plt.savefig('optimized_c45_decision_tree.png', dpi=300, bbox_inches='tight',
284               facecolor='white', edgecolor='none')
285     plt.show()
286
287     # 输出准确率
288     print(f"\nC4.5决策树准确率: {accuracy:.4f}")
289
290
291 if __name__ == "__main__":
292     main()
```

## mainCART.py

```

1  import numpy as np
2  import pandas as pd
3  from mindspore import context
4  import matplotlib.pyplot as plt
5  from sklearn.preprocessing import LabelEncoder
6  from sklearn.metrics import accuracy_score
7
8
9  class OptimizedCARTDecisionTree:
10     def __init__(self, max_depth=5, min_samples_split=2):
11         self.tree = None
12         self.max_depth = max_depth
13         self.min_samples_split = min_samples_split
14
15     def calc_gini(self, y):
16         """ 计算基尼指数 """
17         if len(y) == 0:
18             return 0
19         _, counts = np.unique(y, return_counts=True)
20         probabilities = counts / len(y)
21         return 1 - np.sum(probabilities ** 2)
22
23     def find_best_split(self, X, y):
24         """ 找到最佳分裂特征和分裂点 """
25         best_gain = 0
26         best_feature = None
27         best_threshold = None
28         parent_gini = self.calc_gini(y)
29
30         # 记录所有可能的分裂点
31         potential_splits = []
32
33         for feature in X.columns:
34             # 对于每个特征, 找最佳分裂点
35             unique_values = np.unique(X[feature])
36
37             for threshold in unique_values:
38                 # 连续特征分裂
39                 left_mask = X[feature] <= threshold
40                 right_mask = ~left_mask
41
42                 y_left = y[left_mask]
43                 y_right = y[right_mask]
44
45                 if len(y_left) < self.min_samples_split or len(y_right) < self.min_samples_split:
46                     continue
47
48                 # 加权基尼指数
49                 gini_left = self.calc_gini(y_left)
50                 gini_right = self.calc_gini(y_right)
51                 split_gini = (len(y_left) * gini_left + len(y_right) * gini_right) / len(y)
52
53                 # 基尼增益
54                 gini_gain = parent_gini - split_gini
55
56                 potential_splits.append({
57                     'feature': feature,
58                     'threshold': threshold,
59                     'gain': gini_gain,
60                     'left_mask': left_mask,
61                     'right_mask': right_mask
62                 })
63
64         # 如果没有有效分裂
65         if not potential_splits:
66             return None, None, 0

```



```

67
68     # 选择增益最大的分裂
69     best_split = max(potential_splits, key=lambda x: x['gain'])
70
71     return (best_split['feature'],
72           best_split['threshold'],
73           best_split['gain'],
74           best_split['left_mask'],
75           best_split['right_mask'])
76
77 def _build_tree(self, X, y, depth=0):
78     # 停止条件
79     if (depth >= self.max_depth or
80         len(y) < self.min_samples_split or
81         len(np.unique(y)) == 1):
82         # 返回出现最多的类别
83         most_common = np.bincount(y).argmax()
84         return {'prediction': most_common}
85
86     # 寻找最佳分裂
87     split_result = self.find_best_split(X, y)
88
89     # 无法分裂
90     if split_result[0] is None:
91         most_common = np.bincount(y).argmax()
92         return {'prediction': most_common}
93
94     # 解包分裂结果
95     best_feature, best_threshold, gain, left_mask, right_mask = split_result
96
97     # 合并相似的子节点
98     y_left = y[left_mask]
99     y_right = y[right_mask]
100
101     # 检查子节点是否可以合并
102     left_majority = np.bincount(y_left).argmax()
103     right_majority = np.bincount(y_right).argmax()
104
105     # 如果子节点类别相同, 直接返回多数类
106     if left_majority == right_majority:
107         return {'prediction': left_majority}
108
109     # 构建树
110     tree = {
111         'feature': best_feature,
112         'threshold': best_threshold,
113         'gain': gain
114     }
115
116     # 递归构建子树
117     tree['left'] = self._build_tree(
118         X[left_mask], y[left_mask], depth + 1
119     )
120     tree['right'] = self._build_tree(
121         X[right_mask], y[right_mask], depth + 1
122     )
123
124     return tree
125
126 def fit(self, X, y):
127     self.tree = self._build_tree(X, y)
128     return self
129
130 def predict(self, X):
131     return np.array([self._predict_single(row, self.tree) for _, row in X.iterrows()])
132
133 def _predict_single(self, x, tree):
134     if 'prediction' in tree:
135         return tree['prediction']
136

```

```

137     feature_value = x[tree['feature']]
138     if feature_value <= tree['threshold']:
139         return self._predict_single(x, tree['left'])
140     else:
141         return self._predict_single(x, tree['right'])
142
143
144 def plot_cart_tree(tree, ax):
145     """绘制完整的CART决策树"""
146     decision_node_color = '#B3E5FC' # 浅蓝色决策节点
147     leaf_node_high_color = '#C8E6C9' # 浅绿色高销量节点
148     leaf_node_low_color = '#FFCDD2' # 浅红色低销量节点
149
150     def draw_node(x, y, content, node_type='decision'):
151         """绘制节点"""
152         if node_type == 'decision':
153             color = decision_node_color
154             size = 2000
155         else:
156             color = leaf_node_high_color if node_type == 'high' else leaf_node_low_color
157             size = 1500
158
159         ax.scatter(x, y, s=size, c=color, edgecolor='black', alpha=0.6, zorder=2)
160         ax.text(x, y, content, ha='center', va='center', color='black',
161                fontsize=12, fontweight='bold')
162
163     def draw_connection(start, end, text=''):
164         """绘制连接线"""
165         ax.plot([start[0], end[0]], [start[1], end[1]], 'black',
166                linestyle='--', zorder=1)
167         if text:
168             mid_x = (start[0] + end[0]) / 2
169             mid_y = (start[1] + end[1]) / 2
170             ax.text(mid_x, mid_y + 0.1, text, ha='center', va='bottom',
171                    color='black', fontsize=10)
172
173     def draw_tree(node, x, y, dx):
174         """递归绘制树"""
175         if 'prediction' in node:
176             # 叶节点
177             value = "HIGH" if node['prediction'] == 1 else "LOW"
178             node_type = 'high' if node['prediction'] == 1 else 'low'
179             draw_node(x, y, f"Sales:\n{value}", node_type)
180             return
181
182         # 决策节点
183         feature_name = node['feature']
184         threshold = node['threshold']
185         gini = node['gain']
186         draw_node(x, y, f"{feature_name}<={threshold}\nGini={gini:.3f}")
187
188         # 递归绘制左子树
189         if 'left' in node:
190             next_x = x - dx
191             next_y = y - 1
192             draw_connection((x, y), (next_x, next_y), 'Yes')
193             draw_tree(node['left'], next_x, next_y, dx / 2)
194
195         # 递归绘制右子树
196         if 'right' in node:
197             next_x = x + dx
198             next_y = y - 1
199             draw_connection((x, y), (next_x, next_y), 'No')
200             draw_tree(node['right'], next_x, next_y, dx / 2)
201
202     # 设置图形属性
203     ax.set_title('CART Decision Tree', pad=20, size=14)
204     ax.set_xlim(-6, 6)
205     ax.set_ylim(0, 5)
206     ax.axis('off')

```

```
207
208     # 从根节点开始递归绘制整棵树
209     draw_tree(tree, 0, 4, 3)
210
211 def main():
212     # 读取数据
213     data = pd.read_csv('ex3dataEn.csv', header=None,
214                       names=['weather', 'weekend', 'promotion', 'sales'])
215
216     # 数据预处理
217     le = LabelEncoder()
218     for column in data.columns:
219         data[column] = le.fit_transform(data[column])
220
221     X = data.iloc[:, :3]
222     y = data.iloc[:, 3]
223
224     # 训练CART决策树
225     cart_tree = OptimizedCARTDecisionTree()
226     cart_tree.fit(X, y)
227
228     # 预测和计算准确率
229     y_pred = cart_tree.predict(X)
230     accuracy = accuracy_score(y, y_pred)
231
232     # 创建决策树可视化
233     fig, ax = plt.subplots(figsize=(12, 8))
234     plot_cart_tree(cart_tree.tree, ax)
235
236     plt.savefig('optimized_cart_decision_tree.png', dpi=300, bbox_inches='tight',
237               facecolor='white', edgecolor='none')
238     plt.show()
239
240     # 输出准确率
241     print(f"\nCART决策树准确率: {accuracy:.4f}")
242
243
244 if __name__ == "__main__":
245     main()
```