决策树算法(ID3)

实验报告

计算机科学与技术学院
2352018 刘彦
2024 年 12 月 5 日

一、 实验目的

1. 理解和掌握决策树算法

通过实现 ID3 算法，理解决策树的基本原理，包括信息增益、熵的计算，以及如何通过特征选择和数据划分来构建决策树。

2. 实现决策树的训练和预测功能

实现决策树的训练过程（即通过数据训练决策树模型），并能够利用训练好的模型进行新样本的预测，掌握如何将决策树应用于实际问题。

3. 后剪枝技术的应用

通过实验理解和应用决策树的后剪枝技术，学习如何通过减少树的复杂度来提高模型的泛化能力，避免过拟合。

二、 实验内容

1. 数据准备

使用 pandas 从 Excel 文件中加载数据集，包括训练数据（data.xlsx）、验证数据（validation_data.xlsx）以及样本数据（samples.xlsx）。确保数据格式正确无误，包含所需特征和目标列。

2. 决策树构建

我采用递归的方法构建决策树，基于输入的数据集及其特征选择最佳特征分裂数据，并在满足终止条件时生成叶节点。

①终止条件判断

- 纯度终止：如果数据集中目标变量的所有值都相同，则直接返回该值作为叶节点。
- 无特征可用或深度限制：如果没有剩余特征可用，或者已达到最大深度，返回目标变量的众数作为叶节点。
- 样本数不足：如果样本数量少于设定的最小分裂数，停止分裂，返回目标变量的众数。
- 信息增益不足：如果最佳分裂特征的信息增益小于阈值，停止分裂，返回目标变量的众数。

②特征选择

调用 select_best_feature 函数，基于最大信息增益选择当前最优的分裂特征。

③构建子树

- 根据最佳特征的所有可能取值，划分数据集为多个子集。
- 对每个子集递归调用 build_tree 函数构建子树。
- 将构建好的子树添加到当前特征的分支中，最终形成一棵完整的决策树。

④返回结果

返回值是一个嵌套的字典结构，表示决策树：键为特征名称。值为子树或叶节点。

三、 实验过程

1. 实验环境准备

确保已安装 python 环境，并配置必要的依赖库（如 pandas、math 等）。准备好实验所需的 excel 数据文件：

- data.xlsx：用于训练决策树的训练数据。
- validation_data.xlsx：用于决策树剪枝的验证数据。
- samples.xlsx：用于预测的样本数据。

2. 加载数据

使用 load_data_from_excel()函数加载数据文件。

确保数据以字典形式存储，验证数据格式是否正确（例如，检查特征列是否齐全，目标列是否一致）。

```
# 加载训练和验证数据
training_data = load_data_from_excel('data.xlsx')
validation_data = load_data_from_excel('validation_data.xlsx')
sample_data = load_data_from_excel('samples.xlsx')
```

3. 决策树模型构建

创建 DecisionTreeID3 类的实例化对象。

调用 fit()函数，根据训练数据构建决策树。

设置相关参数（如 max_depth, min_samples_split, min_gain），观察其对决策树复杂度的影响。

```
# 定义特征和目标
features = ['Age', 'Study Time', 'Attendance', 'Participation', 'Homework Hours',
'Previous Grades']
target = 'Pass'
# 创建并拟合决策树
tree = DecisionTreeID3()
tree.fit(training_data, features, target, min_samples_split=5, min_gain=0.05)
```

4. 打印决策树结构

使用 print_tree() 函数，打印生成的决策树，直观展示树的层级结构和分裂依据。

```
# 打印生成的决策树结构
print("生成的决策树结构:")
print_tree(tree.tree)
```

5. 决策树剪枝

使用验证数据进行后剪枝，通过 post_prune()函数对决策树进行优化。旨在减少模型的复杂性，提高泛化能力，避免过拟合。剪枝通过简化决策树结构，使其对新数据具有更好的预测性能。

打印剪枝后的决策树，并与未剪枝的树对比。

```
# 后剪枝
tree.tree = tree.post_prune(tree.tree, validation_data, target)
# 打印剪枝后的决策树结构
print("\n剪枝后的决策树结构:")
print_tree(tree.tree)
```

6. 样本数据预测

对加载的样本数据进行预测，调用 predict_with_names()函数输出每个样本的预测结果和对应的名字。

```
print("\n样本数据预测结果:")
# 对样本数据进行预测并打印结果
for sample in sample_data:
    predict_with_names(tree, sample)
```

7. 性能评估与参数优化

调整训练时的参数（如 min_samples_split， min_gain），记录模型性能的变化趋势，比较剪枝前后决策树在验证数据集上的性能（如分类准确率）。调整参数具体影响和优化调整在后续内容中说明，并进行实验结果记录与分析。

8. 实验执行逻辑

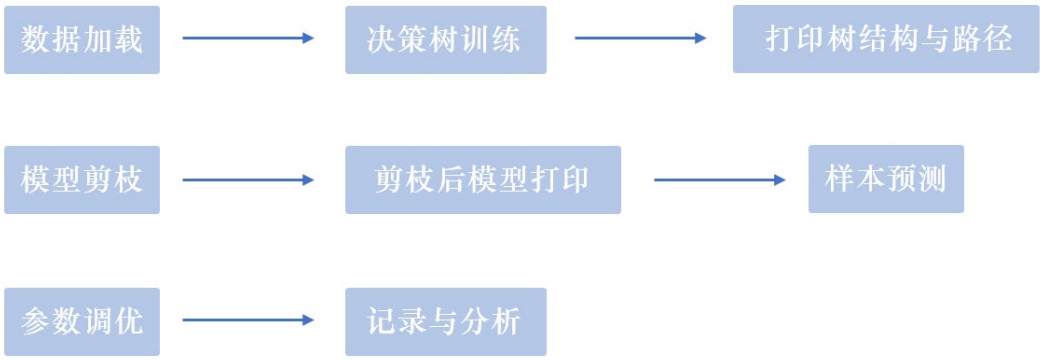

图 1 实验执行逻辑图

四、 测试样例说明

1. 数据来源

测试样例的数据集来源于人工构造的学生学业表现数据集，其中包括多种与学习成绩相关的特征。这些特征的选择是为了模拟不同情境下的学生学习习惯和行为，并通过决策树分析其与成绩通过与否之间的关系。

2. 数据准备

该数据集模拟了一个学生考试通过情况的数据集，包括：学生的年龄段（Age）、学习时间（Study Time）、课堂出勤率（Attendance）、课堂参与度（Participation）、每天完成作业的时间（Homework Hours）、先前成绩等级（Previous Grades），以及目标变量是否通过考试（Pass）。数据涵盖多种学习习惯和学业表现组合，适合用于训练和验证决策树模型。

3. 实验目标

通过上述测试样例，观察模型在多种特征组合下的预测表现，评估模型能否准确识别学生通过与否的模式，不同特征对决策的贡献程度（如学习时间、历史

成绩是否对目标变量有更大影响），和模型在面对极端或异常数据时的分类能力。

4. 数据集示例

下面是 data.xlsx 的一种可能的情况，也是文件夹中的 data.xlsx：

Age	Study Time	Attendance	Participation	Homework Hours	Previous Grades	Pass
Youth	High	High	Active	2	A	Y
Youth	High	Medium	Passive	1	B	Y
Youth	Medium	Medium	Active	2	A	Y
Youth	Medium	Low	Passive	3	C	N
Youth	Low	Medium	Active	3	B	Y
Youth	Low	Low	Passive	1	C	N
Youth	High	Medium	Active	2	C	Y
Youth	Medium	Medium	Active	2	C	Y
Youth	Medium	Medium	Active	3	B	Y
Youth	Medium	Low	Passive	1	C	N
Middle-aged	High	High	Active	3	A	Y
Middle-aged	High	Medium	Active	1	B	Y
Middle-aged	Medium	Medium	Passive	2	A	Y
Middle-aged	Medium	Low	Passive	1	C	N
Middle-aged	Low	Medium	Active	1	B	N
Middle-aged	Low	Low	Passive	1	C	N
Middle-aged	Medium	Medium	Active	2	B	Y
Middle-aged	Medium	Medium	Passive	1	C	N
Senior	Low	Medium	Active	3	B	Y
Senior	Low	Low	Passive	1	C	N
Senior	Medium	Medium	Active	2	B	N
Senior	Medium	Medium	Passive	1	C	N
Senior	Low	Medium	Passive	1	B	N

表 1 data.xlsx 的内容

下面是 samples.xlsx 的一种可能的情况，也是文件夹中的 samples.xlsx：

Name	Age	Study Time	Attendance	Participation	Homework Hours	Previous Grades
Alice	Youth	Low	Low	Passive	0.5	C
Sam	Senior	High	High	Active	3	B
Li	Middle-aged	Medium	High	Active	2	B
Bob	Middle-aged	Medium	Medium	Passive	1	C
Charlie	Youth	Medium	Low	Active	2	C
Diana	Youth	High	High	Active	3	A
Edward	Middle-aged	Medium	Medium	Passive	1	B
Fiona	Senior	Medium	Low	Active	2	C
Zhang	Youth	Medium	High	Active	2	A

表 2 samples.xlsx 的内容

五、 测试结果及分析

1. 测试结果

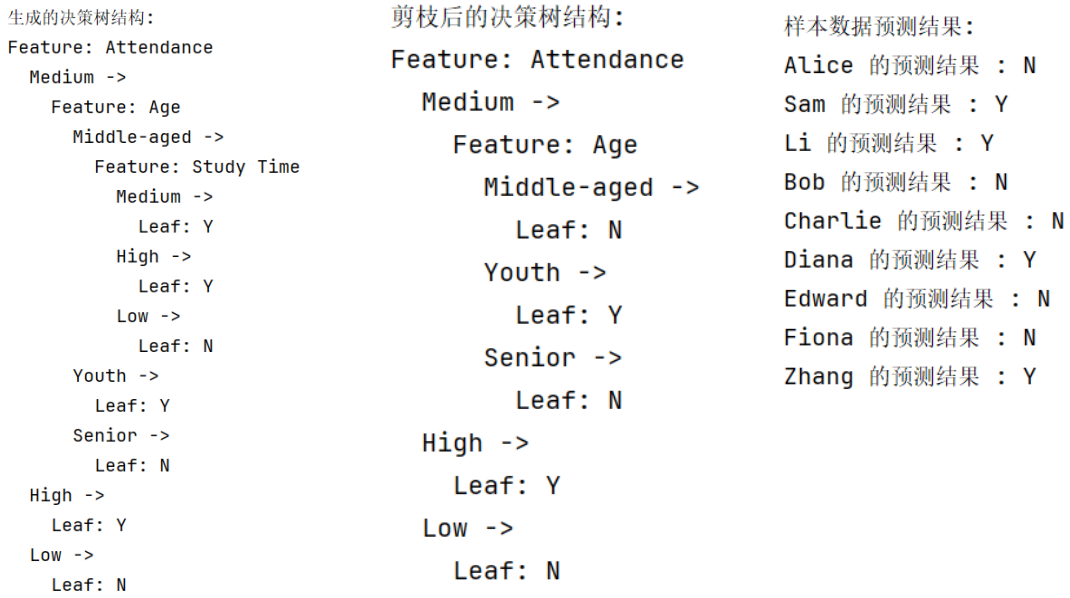

图 2 (min_samples_split,min_gain)=(5,0.05)的实验结果

在 min_samples_split 和 min_gain 均取常见值(5;0.05)时,从测试结果可以看出，模型对样例数据的预测准确率很高。这表明决策树模型在当前数据集上的分类效果非常好，能够准确地捕捉特征与目标变量之间的关系。

2. 特征影响分析

①学习时间（Study Time）和历史成绩（Previous Grades）

测试样例中,学习时间和历史成绩对最终通过与否的预测有较大影响。例如，学习时间高且历史成绩为 A 的样本大多预测为通过，这与实际结果一致。

②课堂参与度（Participation）和出勤率（Attendance）

积极参与课堂和高出勤率的样例也更倾向于被预测为通过,表明这些特征对决策树的分裂贡献较大。

3. 实验结果分析

①极端情况分析

samples.xlsx 中包含一些极端情况（如作业时间很短或成绩较差），模型成功地预测了这些样本的结果，说明模型具有一定的鲁棒性。

②剪枝效果分析

剪枝后的决策树在模型复杂度和预测性能之间达到了较好的平衡，去除了冗余分支，避免了过拟合现象。通过对比剪枝前后的准确性，剪枝并未对测试结果产生负面影响。

③局限性分析

当前数据集规模较小，样本数量和特征分布有限，可能导致模型在更大规模或更复杂的数据集上表现下降。

4. 参数的调试

说明：在测试其中一个参数时，其他参数均取常见值。(min_samples_split, min_gain)=(5,0.05)的情况前文已经列出，用来做对照。

①min_samples_split 的调试

该参数指定了分裂内部节点所需的最小样本数。如果一个节点的样本数少于 min_samples_split，则不会对该节点进行分裂，而是将其作为叶节点。

a) min_samples_split= 3

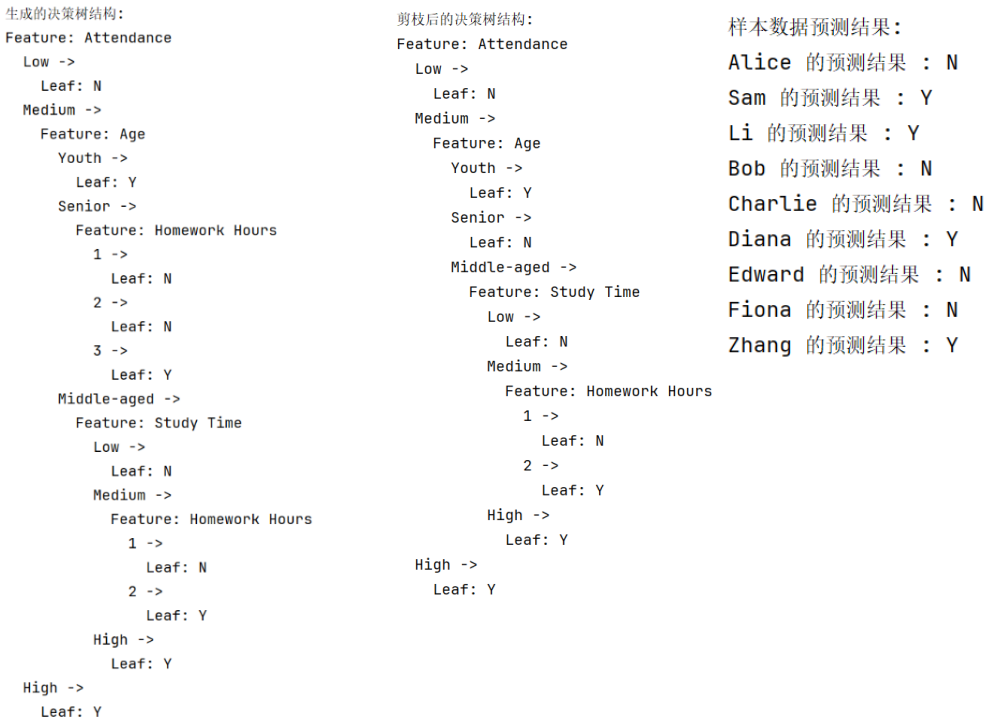

图 3 (min_samples_split,min_gain)=(3,0.05)的实验结果

较低的值：允许更多的分裂，生成更复杂的树，可能会更好地拟合训练数据，但也有可能导致过拟合，尤其是当树过于复杂并学习了数据中的噪声时。

b) min_samples_split= 8

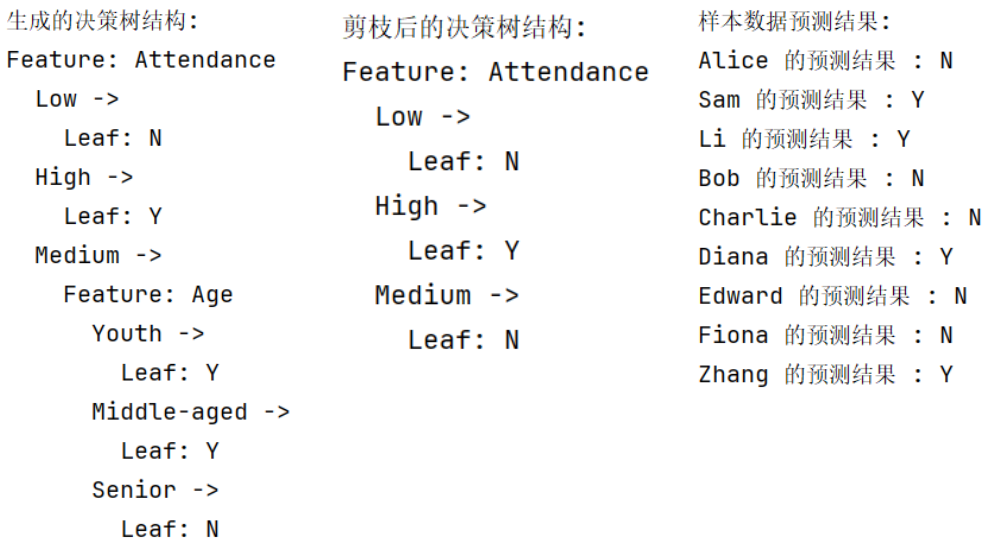

图 4 (min_samples_split,min_gain)=(8,0.05)的实验结果

较高的值：增加了分裂节点所需的最小样本数，这样会生成较简单的树，减少过拟合的风险，确保只有在有足够样本的情况下才进行分裂。此时的值已经出现了一些偏差。

c) min_samples_split= 15

图 5 (min_samples_split,min_gain)=(15,0.05)的实验结果

过高的值会带来错误的结果。

②min_gain 的调试

该参数用于控制树的分裂过程，避免过拟合。当分裂节点时，如果某个特征

的分裂所带来的信息增益小于 min_gain，则停止分裂。通过设置 min_gain，可以避免模型对训练数据中过于细微的差异做出复杂决策，减少过拟合的风险。

a) min_gain= 0.01

```
生成的决策树结构:
Feature: Attendance
  Low ->
    Leaf: N
  Medium ->
    Feature: Age
      Youth ->
        Leaf: Y
      Middle-aged ->
        Feature: Study Time
          Low ->
            Leaf: N
          Medium ->
            Leaf: Y
          High ->
            Leaf: Y
      Senior ->
        Leaf: N
  High ->
    Leaf: Y
```

```
剪枝后的决策树结构:
Feature: Attendance
  Low ->
    Leaf: N
  Medium ->
    Feature: Age
      Youth ->
        Leaf: Y
      Middle-aged ->
        Leaf: N
      Senior ->
        Leaf: N
  High ->
    Leaf: Y
```

```
样本数据预测结果:
Alice 的预测结果 : N
Sam 的预测结果 : Y
Li 的预测结果 : Y
Bob 的预测结果 : N
Charlie 的预测结果 : N
Diana 的预测结果 : Y
Edward 的预测结果 : N
Fiona 的预测结果 : N
Zhang 的预测结果 : Y
```

图 6 (min_samples_split,min_gain)=(15,0.01)的实验结果

较低的 min_gain 值允许更复杂的树结构，但可能导致过拟合。

b) min_gain= 0.5

```
生成的决策树结构:
Leaf: Y

剪枝后的决策树结构:
Leaf: Y

样本数据预测结果:
Alice 的预测结果 : Y
Sam 的预测结果 : Y
Li 的预测结果 : Y
Bob 的预测结果 : Y
Charlie 的预测结果 : Y
Diana 的预测结果 : Y
Edward 的预测结果 : Y
Fiona 的预测结果 : Y
Zhang 的预测结果 : Y
```

图 7 (min_samples_split,min_gain)=(15,0.5)的实验结果

较高的 min_gain 值会限制分裂数量，生成更简单的决策树。过高的值会带来错误的结果。

六、 附录：主要源代码

```
import math
import pandas as pd
from collections import Counter
class DecisionTreeID3:
    def __init__(self):
        self.tree = None  # 存储生成的决策树
    def fit(self, data, features, target, max_depth=None, current_depth=0,
min_samples_split=2, min_gain=0.01):
        self.tree = self.build_tree(data, features, target, max_depth,
current_depth, min_samples_split, min_gain)

    def build_tree(self, data, features, target, max_depth, current_depth,
min_samples_split, min_gain):
        # 终止条件
        if len(set(row[target] for row in data)) == 1:
            return data[0][target]
        if not features or (max_depth is not None and current_depth >=
max_depth):
            return Counter(row[target] for row in data).most_common(1)[0][0]
        # 如果样本数少于最小分裂数，停止分裂
        if len(data) < min_samples_split:
            return Counter(row[target] for row in data).most_common(1)[0][0]
        # 选择最佳特征
        best_feature = self.select_best_feature(data, features, target)
        best_gain = self.calculate_info_gain(data, best_feature, target)
        # 如果信息增益小于阈值，停止分裂
        if best_gain < min_gain:
            return Counter(row[target] for row in data).most_common(1)[0][0]
        # 构建子树
        tree = {best_feature: {}}
        feature_values = set(row[best_feature] for row in data)
        for value in feature_values:
            subset = [row for row in data if row[best_feature] == value]
            subtree = self.build_tree(subset, [f for f in features if f !=
best_feature], target, max_depth, current_depth + 1, min_samples_split,
min_gain)
            tree[best_feature][value] = subtree
        return tree
    def select_best_feature(self, data, features, target):
        base_entropy = self._entropy([row[target] for row in data])
        best_gain = 0
        best_feature = None
```

```
        for feature in features:
            info_gain = self.calculate_info_gain(data, feature, target)
            if info_gain > best_gain:
                best_gain = info_gain
                best_feature = feature
        return best_feature
    def calculate_info_gain(self, data, feature, target):
        base_entropy = self._entropy([row[target] for row in data])
        feature_values = [row[feature] for row in data]
        subsets = {value: [row[target] for row in data if row[feature] ==
value] for value in set(feature_values)}
        new_entropy = sum((len(subset) / len(data)) * self._entropy(subset)
for subset in subsets.values())
        return base_entropy - new_entropy
    @staticmethod
    def _entropy(values):
        counts = Counter(values)
        total = len(values)
        return -sum((count / total) * math.log2(count / total) for count in
counts.values() if count > 0)
    def predict(self, sample):
        return self._classify(sample, self.tree)
    def _classify(self, sample, tree):
        if not isinstance(tree, dict):
            return tree
        feature, branches = next(iter(tree.items()))
        feature_value = sample.get(feature)
        if feature_value not in branches:
            return None
        return self._classify(sample, branches[feature_value])
    def post_prune(self, tree, data, target):
        def prune(subtree, subset):
            if not isinstance(subtree, dict):
                return subtree
            feature = next(iter(subtree))
            branches = subtree[feature]
            for value, branch in branches.items():
                branch_data = [row for row in subset if row[feature] == value]
                branches[value] = prune(branch, branch_data)
            combined_label = Counter(row[target] for row in
subset).most_common(1)[0][0]
            if all(not isinstance(branch, dict) for branch in
branches.values()):
                leaf_predictions = [branches[row[feature]] for row in subset if
```

```
row[feature] in branches]
                error_before = sum(1 for pred, actual in zip(leaf_predictions,
[row[target] for row in subset]) if pred != actual)
                error_after = sum(1 for actual in [row[target] for row in
subset] if actual != combined_label)
                if error_after <= error_before:
                    return combined_label
            return subtree
        return prune(tree, data)
# 从 Excel 文件中读取数据
def load_data_from_excel(file_path):
    data = pd.read_excel(file_path,
engine='openpyxl').to_dict(orient='records')
    return data
# 打印决策树结构
def print_tree(tree, indent=""):
    if not isinstance(tree, dict):
        print(indent + "Leaf:", tree)
        return
    for feature, branches in tree.items():
        print(indent + f"Feature: {feature}")
        for value, subtree in branches.items():
            print(indent + f"  {value} ->")
            print_tree(subtree, indent + "    ")
def predict_with_names(tree, sample):
    name = sample.get('Name')  # 获取名字
    prediction = tree.predict(sample)  # 使用决策树预测结果
    print(f"{name} 的预测结果 : {prediction}")
    return prediction
if __name__ == "__main__":
    略，具体实现放在【三、实验过程的 2-6】中
```