**《机器学习基础》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2022级计算机科学与技术（卓越）01** | | | **姓名** |  |
| **实验题目** | **决策树算法实践** | | | | | |
| **实验时间** | **2024/05/19** | | **实验地点** | **DS3401** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 □综合性** | | |
| 教师评价：  **□**算法/实验过程正确； **□**源程序/实验内容提交 **□**程序结构/实验步骤合理；  **□**实验结果正确； **□**语法、语义正确； **□**报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  掌握决策树回归算法原理。 | | | | | | |
| 二、实验项目内容  1.理解并**描述**决策树分类、回归算法原理。  2.**编程**实践，将决策树分类、回归算法分别应用于合适的数据集(如鸢尾花、波士顿房价预测、UCI数据集、Kaggle数据集)，要求算法至少用于两个数据集(分类2个，回归2个)。 | | | | | | |
| 三、实验过程或算法（源程序）  1. 决策树模型  决策树模型基于树状结构（如图），通过特征选择来模拟人脑的决策 行为，最终实现样本的分类任务。  IMG_256  决策树的主要任务在于生成树的过程：通过一定的标准每次选择一个 属性，作为当前结点的分裂依据，将选定属性的不同取值作为该结点的子 结点。重复上述操作从上至下递归生成一棵树，直到数据集不可分。需要 注意，由于每个属性并不一定只有两种取值，因此决策树并不一定是二叉 树结构。决策树的基本算法实现如下图所示：   1. 划分选择   决策树模型在生成树时首先就是要通过一定的标准每次选择一个属性，作为当前结点的分裂依据，也就是选择最优划分属性。  一般而言，随着划分过程不断进行，希望决策树的分支结点所包含的样本尽可能属于同一类别，即纯度较高。比较经典的划分方法有信息增益 ID3、信息增益率C4.5 和基尼指数 CART。但是，ID3 决策树学习算法对可取值数目较多的属性有所偏好； C4.5决策树学习算法对可取值数目较少的属性有所偏好，泛化性能都略低。因此本实验采用 CART 决策树，即基于“基尼指数”来选择划分属性。  基尼指数越小，说明以该属性为划分的数据集纯度越高。 设某个离散属性 a 有 V 个可能的取值{a1 ,a2 ,...,aV}，那么如果用 a 作为划分标准，就会有 V 个分支结点。设 D 表示当前样本的集合，则 Dv表示 D 中以 a 为划分的所有属性值为 av的样本集合（1<=v<=V）。由此可计算 出数据集 D 上以属性 a 为划分的基尼指数为：   1. 连续值处理   前面介绍的模型中讨论的属性都是离散的，但本次实验需要处理的属 性取值都是连续的，因此需要将其离散化处理，才能用于前面介绍的 CART 决策树。连续属性离散化采用二分思想，取样本在某属性上取值的升序序 列的相邻取值的中点作为待考察的划分点，依次计算该属性下每个待考察 的划分点的基尼指数，最小的就是该属性可选取的最优二叉划分点。通过 比较不同属性的最优划分点的基尼指数，最终确定当前结点的分裂标准 （属性和分裂值）。  需要注意的是，连续属性在划分后在其子节点中仍然可以作为划分，形如下图所示  IMG_256   1. 决策树回归算法   上述过程介绍的决策树主要用于分类，若要将其用于回归，只需要在 特征选择处做出改进：分类树是通过计算最小基尼值确定最优划分属性及 其分界值，但回归模型中不存在分类标签一说，因此也不存在基尼值的概 念。 对于不断二分的样本集，每划分出来的空间 Rj（i=1, 2）中取该空间内样本回归值的均值作为该空间的回归值，即  损失函数RSS：  是回归树在特征选择时，选取 RSS 最小的属性及其分界值即可。其 他建树过程与分类树一致，最后采用 R2作为性能衡量指标:  其中分子表示真实值与预测值的差的平方和，分母表示真实值与均值的差的平方和。R2的取值在负无穷到 1 之间，结果越小说明模型拟合效果越差，越接近1说明模型拟合效果越好。  决策树实验代码：   1. import pandas as pd 2. import numpy as np 3. *#采用ID3决策树作为分类器分类西瓜数据集* 4. *#计算信息熵函数* 5. def cal\_information\_entropy(data): 6. *#选取样本集最后一列* 7. data\_label = data.iloc[:,-1] 8. *#  Series：{标签，数量}* 9. label\_class =data\_label.value\_counts() 10. Ent = 0 11. for k in label\_class.keys(): 12. p\_k = label\_class[k]/len(data\_label) 13. Ent += -p\_k\*np.log2(p\_k) 14. return Ent 15. *#计算当前特征a的信息增益* 16. def cal\_information\_gain(data, a): 17. *#计算整个数据集的信息熵* 18. Ent = cal\_information\_entropy(data) 19. feature\_class = data[a].value\_counts() 20. gain = 0 21. *# 用算熵的公式计算信息增益* 22. for v in feature\_class.keys(): 23. weight = feature\_class[v]/data.shape[0] 24. Ent\_v = cal\_information\_entropy(data.loc[data[a] == v]) *#选择a属性取值为v的行去计算信息熵* 25. gain += weight \* Ent\_v 26. return Ent - gain 27. *#挑选最优特征，即信息增益最大(就是指分一次类之后能让它变得更纯净，比如分成0000 和 1111就是最好的)的特征* 28. def get\_best\_feature(data): 29. *#选取除去最后一列的所有列的名称* 30. features = data.columns[:-1] 31. res = {} *# 用于存储每个特征的信息增益* 32. for a in features: 33. res[a] = cal\_information\_gain(data, a) *# 计算当前特征的信息增益* 34. *# 按照信息增益从大到小排序* 35. res = sorted(res.items(),key=lambda x:x[1],reverse=True) 36. return res[0][0] 37. def drop\_exist\_feature(data, best\_feature): 38. *# 获取最优特征的所有属性值* 39. attr = pd.unique(data[best\_feature]) 40. *# 按照最优特征的属性值划分数据集* 41. new\_data = [(nd, data[data[best\_feature] == nd]) for nd in attr] 42. *# 删除new\_data中最优特征那一列* 43. new\_data = [(n[0], n[1].drop([best\_feature], axis=1)) for n in new\_data] 44. return new\_data 45. *#获取标签最多的那一类* 46. def get\_most\_label(data): 47. data\_label = data.iloc[:,-1] 48. label\_sort = data\_label.value\_counts(sort=True) 49. return label\_sort.keys()[0] 50. *#创建决策树* 51. def create\_tree(data): 52. data\_label = data.iloc[:,-1] 53. *# 如果数据集中只有一种类别，就直接返回这个类别* 54. if len(data\_label.value\_counts()) == 1: 55. return data\_label.values[0] 57. *# 如果数据集在所有特征上完全一致，返回样本数最多的类别* 58. if all(len(data[i].value\_counts()) == 1 for i in data.iloc[:,:-1].columns): 59. return get\_most\_label(data) 60. best\_feature = get\_best\_feature(data) 61. Tree = {best\_feature:{}} 63. for item in drop\_exist\_feature(data,best\_feature): 64. *#递归调用，直到所有特征都分类完毕* 65. *# item[0]是特征的取值，item[1]是剩下的数据集* 66. Tree[best\_feature][item[0]] = create\_tree(item[1]) 67. return Tree 68. def predict(Tree , test\_data): 69. first\_feature = list(Tree.keys())[0] 70. second\_dict = Tree[first\_feature] 71. input\_first = test\_data.get(first\_feature) 72. input\_value = second\_dict[input\_first] 73. if isinstance(input\_value , dict): 74. class\_label = predict(input\_value, test\_data) 75. else: 76. class\_label = input\_value 77. return class\_label 78. if \_\_name\_\_ == '\_\_main\_\_': 79. data = pd.read\_csv('watermelon.csv') 80. dicision\_Tree = create\_tree(data) 81. test\_data\_1 = {'色泽': '青绿', '根蒂': '蜷缩', '敲声': '浊响', '纹理': '清晰', '脐部': '稍凹', '触感': '硬滑'} 82. result = predict(dicision\_Tree,test\_data\_1) 83. if result == 1: 84. print('好瓜') 85. else: 86. print('坏瓜')   鸢尾花数据集：   1. import pandas as pd 2. import numpy as np 3. from sklearn.preprocessing import LabelEncoder 4. label\_encoder = LabelEncoder() 5. *# 处理数据* 6. def process\_data(filepath): 7. data = pd.read\_csv(filepath) 8. data.drop('Id', axis=1, inplace=True)  *# 删除Id列* 9. data['target'] = label\_encoder.fit\_transform(data['target'])  *# 将分类名称转换为数值* 10. return data 11. *# 计算信息熵* 12. def calculate\_entropy(y): 13. if len(y) == 0: 14. return 0 15. p = np.bincount(y) / len(y) 16. p = p[p > 0] 17. return -np.sum(p \* np.log2(p)) 18. *# 寻找最佳分割点* 19. def best\_split(X, y): 20. best\_gain = 0 *# 信息增益* 21. best\_split = None *# 最佳分割点* 22. *# 对特征值进行排序* 23. sorted\_index = X.argsort() 24. sorted\_x = X.iloc[sorted\_index] 25. unique\_values = sorted\_x.unique() 26. for i in range(1, len(unique\_values)): 27. split\_point = (unique\_values[i] + unique\_values[i - 1]) / 2 28. left\_mask = X <= split\_point 29. right\_mask = X > split\_point 30. left\_entropy = calculate\_entropy(y[left\_mask]) 31. right\_entropy = calculate\_entropy(y[right\_mask]) 32. total\_entropy = (left\_mask.sum() \* left\_entropy + right\_mask.sum() \* right\_entropy) / len(y) 33. gain = calculate\_entropy(y) - total\_entropy 34. if gain > best\_gain: 35. best\_gain = gain 36. best\_split = split\_point 37. return best\_split, best\_gain 38. *# 选择最佳特征* 39. def get\_best\_feature(data): 40. features = data.columns[:-1] 41. res = {} 42. for feature in features: 43. split\_point, gain = best\_split(data[feature], data['target']) 44. res[feature] = (split\_point, gain) 45. best\_feature = max(res, key=lambda x: res[x][1]) 46. return best\_feature, res[best\_feature][0] 47. *# 构建决策树* 48. def create\_tree(data, depth=0, max\_depth=10): 49. if len(data['target'].value\_counts()) == 1: 50. return data['target'].values[0] 51. if depth == max\_depth: 52. return data['target'].mode()[0] 53. best\_feature, split\_point = get\_best\_feature(data) 54. left\_data = data[data[best\_feature] <= split\_point] 55. right\_data = data[data[best\_feature] > split\_point] 56. tree = {best\_feature: {}} 57. tree[best\_feature][f"<= {split\_point}"] = create\_tree(left\_data, depth+1, max\_depth) 58. tree[best\_feature][f"> {split\_point}"] = create\_tree(right\_data, depth+1, max\_depth) 59. return tree 60. def predict(tree, sample): 61. if isinstance(tree, dict):  *# 如果当前节点是一个内部节点* 62. feature\_name = list(tree.keys())[0]  *# 获取当前节点的特征名称* 63. feature\_tree = tree[feature\_name] 64. *# 根据特征值和分割条件确定要遵循的分支* 65. for key, sub\_tree in feature\_tree.items(): 66. *# 提取分割点数值* 67. operator, value = key.split(" ") 68. value = float(value) 69. *# 检查样本中的特征值是否满足分支的条件* 70. if operator == "<=" and sample[feature\_name] <= value or operator == ">" and sample[feature\_name] > value: 71. return predict(sub\_tree, sample) 72. else: 73. return tree  *# 如果是叶节点，则返回预测结果* 74. *# 主函数* 75. if \_\_name\_\_ == '\_\_main\_\_': 76. data = process\_data('Iris-data.csv') 77. decision\_tree = create\_tree(data) 78. test\_data\_1 = {'SepalLengthCm':6.1,'SepalWidthCm':2.8,'PetalLengthCm':4.7,'PetalWidthCm':1.2} 79. prediction = label\_encoder.inverse\_transform([predict(decision\_tree, test\_data\_1)])[0] 80. print(prediction)   回归树实验代码：   1. import pandas as pd 2. import numpy as np 3. *# 加载和预处理数据* 4. def process\_data(filepath): 5. data = pd.read\_csv(filepath) 6. return data 7. *# 计算方差* 8. def cal\_mse(y): 9. if len(y) == 0: 10. return 0 11. mean\_y = np.mean(y) 12. return np.mean((y - mean\_y) \*\* 2) 13. *# 寻找最佳分割点* 14. def best\_split(X, y): 15. best\_mse = float('inf') 16. best\_split = None 17. sorted\_index = X.argsort() 18. sorted\_x = X.iloc[sorted\_index] 19. sorted\_y = y.iloc[sorted\_index] 20. unique\_values = sorted\_x.unique() 21. for i in range(1, len(unique\_values)): 22. split\_point = (unique\_values[i] + unique\_values[i - 1]) / 2 23. left\_mask = X <= split\_point 24. right\_mask = X > split\_point 25. mse\_left = cal\_mse(sorted\_y[left\_mask]) 26. mse\_right = cal\_mse(sorted\_y[right\_mask]) 27. total\_mse = (left\_mask.sum() \* mse\_left + right\_mask.sum() \* mse\_right) / len(y) 28. if total\_mse < best\_mse: 29. best\_mse = total\_mse 30. best\_split = split\_point 31. return best\_split, best\_mse 32. def get\_best\_feature(data): 33. features = data.columns[:-1] 34. res = {} 35. for feature in features: 36. split\_point, gain = best\_split(data[feature], data['MEDV']) 37. res[feature] = (split\_point, gain) 38. best\_feature = min(res, key=lambda x: res[x][1]) 39. return best\_feature, res[best\_feature][0] 40. *# 构建决策树回归* 41. def create(data, depth=0, max\_depth=10): 42. if len(data['MEDV']) <= 1 or depth == max\_depth: 43. return np.mean(data['MEDV'])  *# 返回平均值作为预测结果* 45. best\_feature, split\_point = get\_best\_feature(data) 46. left\_data = data[data[best\_feature] <= split\_point] 47. right\_data = data[data[best\_feature] > split\_point] 48. tree = {best\_feature: {}} 49. tree[best\_feature][f"<= {split\_point}"] = create(left\_data, depth+1, max\_depth) 50. tree[best\_feature][f"> {split\_point}"] = create(right\_data, depth+1, max\_depth) 51. return tree 52. def predict(tree, sample): 53. if isinstance(tree, dict):  *# 如果当前节点是一个内部节点* 54. feature\_name = list(tree.keys())[0]  *# 获取当前节点的特征名称* 55. feature\_tree = tree[feature\_name] 56. *# 根据特征值和分割条件确定要遵循的分支* 57. for key, sub\_tree in feature\_tree.items(): 58. *# 提取分割点数值* 59. operator, value = key.split(" ") 60. value = float(value) 61. *# 检查样本中的特征值是否满足分支的条件* 62. if operator == "<=" and sample[feature\_name] <= value or operator == ">" and sample[feature\_name] > value: 63. return predict(sub\_tree, sample) 64. else: 65. return tree  *# 如果是叶节点，则返回预测结果* 66. *# 主函数* 67. if \_\_name\_\_ == '\_\_main\_\_': 68. data = process\_data('housing.csv')  *# 修改为正确的文件路径* 69. decision\_tree = create(data) 70. *# 测试数据应与您的数据集匹配* 71. test\_data = { 72. 'CRIM': 0.7258, 73. 'ZN': 0.0, 74. 'INDUS': 8.14, 75. 'CHAS': 0, 76. 'NOX': 0.538, 77. 'RM': 5.727, 78. 'AGE': 69.5, 79. 'DIS': 3.7965, 80. 'RAD': 4, 81. 'TAX': 307.0, 82. 'PIRATIO': 21.0, 83. 'B': 390.95, 84. 'LSTAT': 11.28 85. } 86. prediction = predict(decision\_tree, test\_data) 87. print("Predicted value:", prediction)   红酒数据集：   1. import pandas as pd 2. import numpy as np 3. *# 加载和预处理数据* 4. def process\_data(filepath): 5. data = pd.read\_csv(filepath) 6. return data 7. *# 计算方差* 8. def cal\_mse(y): 9. if len(y) == 0: 10. return 0 11. mean\_y = np.mean(y) 12. return np.mean((y - mean\_y) \*\* 2) 13. *# 寻找最佳分割点* 14. def best\_split(X, y): 15. best\_mse = float('inf') 16. best\_split = None 17. sorted\_index = X.argsort() 18. sorted\_x = X.iloc[sorted\_index] 19. sorted\_y = y.iloc[sorted\_index] 20. unique\_values = sorted\_x.unique() 21. for i in range(1, len(unique\_values)): 22. split\_point = (unique\_values[i] + unique\_values[i - 1]) / 2 23. left\_mask = X <= split\_point 24. right\_mask = X > split\_point 25. mse\_left = cal\_mse(sorted\_y[left\_mask]) 26. mse\_right = cal\_mse(sorted\_y[right\_mask]) 27. total\_mse = (left\_mask.sum() \* mse\_left + right\_mask.sum() \* mse\_right) / len(y) 28. if total\_mse < best\_mse: 29. best\_mse = total\_mse 30. best\_split = split\_point 31. return best\_split, best\_mse 32. def get\_best\_feature(data): 33. features = data.columns[:-1] 34. res = {} 35. for feature in features: 36. split\_point, gain = best\_split(data[feature], data['Class']) 37. res[feature] = (split\_point, gain) 38. best\_feature = min(res, key=lambda x: res[x][1]) 39. return best\_feature, res[best\_feature][0] 40. *# 构建决策树回归* 41. def create(data, depth=0, max\_depth=10): 42. if len(data['Class']) <= 1 or depth == max\_depth: 43. return np.mean(data['Class'])  *# 返回平均值作为预测结果* 45. best\_feature, split\_point = get\_best\_feature(data) 46. left\_data = data[data[best\_feature] <= split\_point] 47. right\_data = data[data[best\_feature] > split\_point] 48. tree = {best\_feature: {}} 49. tree[best\_feature][f"<= {split\_point}"] = create(left\_data, depth+1, max\_depth) 50. tree[best\_feature][f"> {split\_point}"] = create(right\_data, depth+1, max\_depth) 51. return tree 52. def predict(tree, sample): 53. if isinstance(tree, dict):  *# 如果当前节点是一个内部节点* 54. feature\_name = list(tree.keys())[0]  *# 获取当前节点的特征名称* 55. feature\_tree = tree[feature\_name] 56. *# 根据特征值和分割条件确定要遵循的分支* 57. for key, sub\_tree in feature\_tree.items(): 58. *# 提取分割点数值* 59. operator, value = key.split(" ") 60. value = float(value) 61. *# 检查样本中的特征值是否满足分支的条件* 62. if operator == "<=" and sample[feature\_name] <= value or operator == ">" and sample[feature\_name] > value: 63. return predict(sub\_tree, sample) 64. else: 65. return tree  *# 如果是叶节点，则返回预测结果* 66. *# 主函数* 67. if \_\_name\_\_ == '\_\_main\_\_': 68. data = process\_data('wine.csv') 69. decision\_tree = create(data) 70. test\_data = { 71. 'Alcohol': 13.5, 72. 'Malic\_Acid': 1.8, 73. 'Ash': 2.36, 74. 'Alcalinity\_of\_Ash': 19.0, 75. 'Magnesium': 102, 76. 'Total\_Phenols': 2.5, 77. 'Flavanoids': 2.9, 78. 'Nonflavanoid\_Phenols': 0.3, 79. 'Proanthocyanins': 1.95, 80. 'Color\_Intensity': 5.2, 81. 'Hue': 1.05, 82. 'OD280/OD315\_of\_Diluted\_Wines': 2.8, 83. 'Proline': 1050 84. } 85. prediction = predict(decision\_tree, test\_data) 86. print("Predicted value:", prediction) | | | | | | |
| 1. 实验结果及分析   我们将决策树代码应用在西瓜集上，由于西瓜集西瓜集的数据样本较少，因此我们决策树的最大深度设置为3层，我们最终得到的正确率在96.6%。  IMG_256  鸢尾花数据集的样本属性较少，只有 4 个属性。为了防止过拟合，将最大深度设置为 4 层。由于样本随机划分带来的不确定性，决策树的构建具有随机性，分裂点的属性、基尼值都有一定的波动性，因此准确率并不稳定，大约在 87%~100%之间，其中93.3%是最普遍的正确率。  下图我将鸢尾花的一个数据的特征输入后得到的结果，可以发现成功将其分类为“Iris-versicolor”  IMG_257IMG_258  房价数据集的样本属性较多，有 11 个属性，包含 20640 个样本，因此训练起来比较费时。将最深层数设为 7，对测试样本进行回归预测的 R2 指数为 0.61，性能表现一般。  IMG_259  对于红酒数据集而言，由于他的样本属性较多，共有 11 个属性。因此决策树层数既不能太大也不能太小，实验中选取层数为 7，最终我们得到测试集准确率为 65.21%。 | | | | | | |
|  | | | | | | |