山东大学 软件 学院

机器学习 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号： 202200201095 | 姓名： 杨伟康 | | 班级： 22网安 |
| 实验题目：决策树分类器在鸢尾花数据集上的实现 | | | |
| 实验学时：4 | | 实验日期： 11.01 | |
| **1.实验目的：**  -深入理解和掌握决策树算法的原理、构建过程以及分类机制，提升综合素养；  -利用决策树分类器实现鸢尾花分类，不仅加深对决策树算法的理解，同时对植物学和相关领域的研究提供有力的技术支持和实际应用价值。 | | | |
| **2.引言：**  **背景和动机**  在机器学习和数据科学领域，决策树作为一种直观且强大的分类和回归方法，因其易于理解和解释、无需复杂的参数调整等特点，受到了广泛的关注和应用。决策树通过递归地选择最优特征进行分割，构建出一个树状结构，实现对样本的分类或回归预测。鸢尾花数据集作为机器学习中的经典数据集，因其数据简单、类别明确，常被用作算法演示和性能测试的基准。本文旨在通过实现决策树分类器在鸢尾花数据集上的应用，验证其分类效果，并探讨其在实际应用中的优势和潜力。  **问题描述**  本文要解决的核心问题是：利用决策树算法对鸢尾花进行分类。在Python的PyCharm环境中，利用sklearn库中的iris数据集，该数据集包含150个样本，每个样本具有花萼长度、花萼宽度、花瓣长度、花瓣宽度四个特征属性。目标是基于这四个特征值，将鸢尾花正确地分类为Iris-Setosa、Iris-Versicolour和Iris-Virginica三类。三类。  **文献综述**  决策树算法通过递归地选择最优特征进行分割，构建出一个树状结构。每个内部节点表示一个特征上的测试，每个分支代表一个测试的结果，而每个叶节点则代表一个类别或回归值。决策树算法具有直观易懂、易于实现、无需复杂的参数调整等优点。在鸢尾花数据集上，决策树算法已被广泛研究和应用，并取得了良好的分类效果。与其他分类算法相比，决策树算法在分类准确率和计算效率上均表现出一定的优势。 | | | |
| **3.实验方法：**  数据集：  Iris鸢尾花数据集（Iris Dataset），包含150个样本，分为三类：Setosa、Versicolor、Virginica。每个样本具有4个特征：花萼长度、花萼宽度、花瓣长度、花瓣宽度。  数据集划分：  将数据集分为训练集（70%）和测试集（30%），或使用交叉验证进行评估。  模型选择：  本实验选择的模型是决策树分类器。决策树分类器通过递归地选择最优特征进行分割，构建出一个树状结构，实现对样本的分类。在本实验中，我们将使用Python的scikit-learn库来实现决策树分类器，并利用其提供的参数调整功能来优化模型的性能。  评估指标  为了评估模型的性能，我们将使用准确率（accuracy）、精确率（precision）、召回率（recall）和F1分数（F1-score）等评估指标。准确率是正确分类的样本数与总样本数的比例；精确率表示被正确分类为正例的样本数与所有被预测为正例的样本数之比；召回率表示被正确分类为正例的样本数与所有实际为正例的样本数之比；F1分数是精确率和召回率的调和平均数，用于综合评估模型的性能。在鸢尾花数据集上，这些评估指标能够全面反映模型的分类效果和性能。。 | | | |
| **4.实验过程报告：**  **模拟决策树分类器**  **import pandas as pd from math import log  # 鸢尾花数据集 from sklearn import datasets # 随机划分训练集和测试集合 from sklearn.model\_selection import train\_test\_split  #选取测试集合比例 rate = 0.15 #处理鸢尾花数据，包括data和target,总共150组数据 def creat\_data():  iris = datasets.load\_iris()  # 划分训练集和测试集  train\_data, test\_data, train\_label, test\_label = \  train\_test\_split(iris.data, iris.target, test\_size=rate)   # 训练集  train\_data = \  pd.DataFrame(train\_data, columns=['sepal length', 'sepal width', 'petal length', 'petal width'])  train\_data['label'] = train\_label   # 测试集  test\_data = \  pd.DataFrame(test\_data, columns=['sepal length', 'sepal width', 'petal length', 'petal width'])  test\_data['label'] = test\_label   return train\_data, test\_data  # 由于鸢尾花的属性都为连续属性，计算attr的均值,作为分类边界 def means(attr, data):  data\_means = data[attr].mean()  return data\_means  #计算熵 def entropy(dataSet):  num\_data = len(dataSet)  labelCount = {}  currentLabels = dataSet['label']   for label in currentLabels:  if label not in labelCount.keys():  labelCount[label] = 1  else:  labelCount[label] += 1   e = 0.0  for key in labelCount:  prob = float(labelCount[key]) / num\_data # 计算单个类的熵值  e -= prob \* log(prob, 2) # 累加每个类的熵值  return e  #对数据按照某个特征进行划分结果 def splitData(attr, data):  mean = means(attr, data)  return data[data[attr] >= mean], data[data[attr] < mean], mean  # 选择最优分类,计算信息增益 def chooseBestFeatureToSplit(data,attrs):  # 此处的attrs属性是数组  e = entropy(data)  max\_gain = 0.0  max\_attr = ''  for attr in attrs:  data1, data2, mean = splitData(attr, data)  e1 = entropy(data1)  e2 = entropy(data2)  gain = e - len(data1)/(0.0+len(data)) \* e1 - len(data2)/(0.0+len(data)) \* e2  if gain > max\_gain:  max\_gain = gain  max\_attr = attr  return max\_attr  #构建决策树 def create\_decision\_tree(data, attrs):  if entropy(data) < 0.1 or len(attrs) == 1:# 如果数据集值剩下了一类，直接返回 # 如果所有特征都已经切分完了，也直接返回  labelCount = {}  currentLabels = data['label']  max = 0  max\_label = ''  for label in currentLabels:  if label not in labelCount.keys():  labelCount[label] = 1  else:  labelCount[label] += 1  if labelCount[label] > max:  max = labelCount[label]  max\_label = label   return max\_label  # 寻找最佳切分的特征  attr = chooseBestFeatureToSplit(data, attrs)  node = {attr: {}}  attrs.remove(attr)  # 递归调用，对每一个切分出来的取值递归建树  data1, data2, mean = splitData(attr, data)  node[attr]["大于等于"+str(mean)] = create\_decision\_tree(data1,attrs)  node[attr]["小于"+str(mean)] = create\_decision\_tree(data2, attrs)  return node   def classify(node, attrs, data):  attr = list(node.keys())[0]# 获取当前节点判断的特征  node = node[attr]  pred = None  for key in node:# 根据特征进行递归  if '大于等于' in key:  k = key  mean = float(str(k).strip('大于等于'))  if data[attr] >= mean:# 如果再往下依然还有子树，那么则递归，否则返回结果  if isinstance(node[key], dict):  pred = classify(node[key], attrs, data)  else:  pred = node[key]  elif '小于' in key:  k = key  mean = float(str(k).strip('小于'))  if data[attr] < mean:  if isinstance(node[key], dict):  pred = classify(node[key], attrs, data)  else:  pred = node[key]  # 如果没有对应的分叉，则找到一个分叉返回  if pred is None:  for key in node:  if not isinstance(node[key], dict):  pred = node[key]  break  return pred   train\_data, test\_data= creat\_data() attrs = ['sepal length', 'sepal width', 'petal length', 'petal width']  tree = create\_decision\_tree(train\_data, attrs) print(tree)  acc = 0.0 for i in range(len(test\_data)):  predict = classify(tree, attrs, test\_data.loc[i])  if predict == test\_data['label'][i]:  acc += 1  print("选取鸢尾花测试集比例为" + str(rate) + "，预测准确率为{:.5f}".format(acc / len(test\_data)))**      **探究决策树不同深度对准确率的影响**  **import numpy as np from sklearn.datasets import load\_iris from sklearn.model\_selection import train\_test\_split import matplotlib.pyplot as plt import matplotlib   # 定义节点类 class Node:  def \_\_init\_\_(self, feature=None, threshold=None, left=None, right=None, value=None):  self.feature = feature # 分割特征  self.threshold = threshold # 分割阈值  self.left = left # 左子树  self.right = right # 右子树  self.value = value # 叶节点取值   # 定义决策树分类器 class DecisionTree:  def \_\_init\_\_(self, max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1):  self.max\_depth = max\_depth # 决策树最大深度  self.min\_samples\_split = min\_samples\_split # 内部节点最小样本数  self.min\_samples\_leaf = min\_samples\_leaf # 叶节点最小样本数  self.root = None # 决策树的根节点   # # 计算基尼指数  # def gini(self, y):  # classes = np.unique(y)  # n\_sample = y.shape[0]  # gini = 0  # for cls in classes:  # gini += (np.sum(y == cls) / n\_sample) \*\* 2  # return 1 - gini   # 计算信息熵  def entropy(self, y):  classes = np.unique(y)  n\_sample = y.shape[0]  entropy = 0  for cls in classes:  ratio = np.sum(y == cls) / n\_sample  entropy -= ratio \* np.log2(ratio + 1e-9)  return entropy   # 计算信息增益  def information\_gain(self, x, y, feature, threshold):  mask = x[:, feature] < threshold  y\_left, y\_right = y[mask], y[~mask]  if len(y\_left) == 0 or len(y\_right) == 0:  return 0  info\_gain = self.entropy(y) - (len(y\_left) / len(y)) \* self.entropy(y\_left) - \  (len(y\_right) / len(y)) \* self.entropy(y\_right)  return info\_gain   # 按照最优特征和阈值分割数据集  def split(self, x, y):  max\_info\_gain = 0  best\_feature, best\_threshold = None, None  n\_sample, n\_feature = x.shape  for feature in range(n\_feature):  thresholds = np.unique(x[:, feature])  for threshold in thresholds:  info\_gain = self.information\_gain(x, y, feature, threshold)  if info\_gain > max\_info\_gain:  max\_info\_gain = info\_gain  best\_feature = feature  best\_threshold = threshold  if best\_feature is None:  return None, None, None  mask = x[:, best\_feature] < best\_threshold  X\_left, y\_left = x[mask], y[mask]  X\_right, y\_right = x[~mask], y[~mask]  return best\_feature, best\_threshold, X\_left, y\_left, X\_right, y\_right   # 计算叶节点取值  def leaf\_value(self, y):  classes, counts = np.unique(y, return\_counts=True)  idx = np.argmax(counts)  return classes[idx]   # 创建决策树  def build\_tree(self, X, y, depth=0):  n\_sample, n\_feature = X.shape  n\_cls = len(np.unique(y))  # 如果样本全部属于同一个类别或深度达到最大深度或样本数量太少则创建叶节点  if n\_cls == 1 or depth == self.max\_depth or n\_sample < self.min\_samples\_split:  leaf\_value = self.leaf\_value(y)  return Node(value=leaf\_value)  # 按照最优特征和阈值分割数据集  feature, threshold, X\_left, y\_left, X\_right, y\_right = self.split(X, y)  # 如果分割后的样本数量太小则创建叶节点  if X\_left is None:  leaf\_value = self.leaf\_value(y)  return Node(value=leaf\_value)  # 如果分割后的样本数量少于叶节点最小样本数则创建叶节点  if len(X\_left) < self.min\_samples\_leaf or len(X\_right) < self.min\_samples\_leaf:  leaf\_value = self.leaf\_value(y)  return Node(value=leaf\_value)  # 创建分割节点  node = Node(feature, threshold)  # 递归创建左右子树  node.left = self.build\_tree(X\_left, y\_left, depth + 1)  node.right = self.build\_tree(X\_right, y\_right, depth + 1)  return node   # 剪枝函数   def prune(self, node, X\_val, y\_val):  if node.left is None and node.right is None:  return False  # 遍历左右子树，并记录下所有叶节点  if node.left and not self.prune(node.left, X\_val, y\_val):  return False  if node.right and not self.prune(node.right, X\_val, y\_val):  return False  # 保存子树，接下来我们将尝试裁剪它  tmp\_left = node.left  tmp\_right = node.right  # 裁剪子树，将其变为一个叶节点  node.left = None  node.right = None  # 计算裁剪前的准确率  y\_pred = self.predict(X\_val)  accuracy\_before = np.mean(y\_pred == y\_val)  # 计算裁剪后的准确率  leaf\_value = self.leaf\_value(y\_val)  y\_pred = [leaf\_value] \* len(y\_val)  accuracy\_after = np.mean(y\_pred == y\_val)  if accuracy\_before <= accuracy\_after:  # 裁剪后的准确率更高，则保留剪枝后的叶节点  node.left = tmp\_left  node.right = tmp\_right  return False  else:  # 裁剪后的准确率更低，则保留未剪枝的节点  return True   # 使用训练数据集创建决策树  def fit(self, X, y):  X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2) # 随机选择20%的数据作为验证集  self.root = self.build\_tree(X\_train, y\_train)  self.prune(self.root, X\_val, y\_val) # 剪枝   # 预测单个样本的类别  def \_predict(self, x, node):  if node.value is not None:  return node.value  if x[node.feature] < node.threshold:  return self.\_predict(x, node.left)  else:  return self.\_predict(x, node.right)   # 剪枝后的预测函数  def predict(self, X):  assert self.root is not None, "build\_tree at first"  y = []  for x in X:  node = self.root  while node.left:  if x[node.feature] < node.threshold:  node = node.left  else:  node = node.right  y.append(node.value)  return np.array(y)   if \_\_name\_\_ == '\_\_main\_\_':  iris = load\_iris()   matplotlib.use('TkAgg')   # 决策树深度的影响  depth = range(3, 13)  accuracies = []  X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.25)   for i in depth:  clf = DecisionTree(max\_depth=i, min\_samples\_split=2, min\_samples\_leaf=1)  clf.fit(X\_train, y\_train)  y\_pred = clf.predict(X\_test)  accuracies.append(np.mean(y\_pred == y\_test))   plt.plot(depth, accuracies)   plt.xlabel("depth")  plt.ylabel("Accuracy")  plt.show()**  ###可视化展示不同深度对决策树准确率的影响 | | | |
| **5. 实验结果与分析**  决策树分类器在鸢尾花数据集上同样展现出了令人满意的性能，准确率较高且易于理解和实现。决策树通过递归地分割特征空间，构建出一个树状模型，能够直观地展示分类过程。在鸢尾花数据集中，花瓣和萼片的长度、宽度等特征被用作分割节点的依据，这些特征在决策树的构建过程中起到了关键作用。由于不同种类的鸢尾花在这些特征上存在显著差异，决策树分类器能够有效地利用这些特征进行分类。然而，决策树的性能也受到一些因素的影响，如树的深度、剪枝策略以及特征的选取。如果树过深，可能会导致过拟合；如果剪枝不足，也可能影响模型的泛化能力。此外，特征的选取和预处理同样对决策树的性能有重要影响。**6. 结论**  决策树分类器在鸢尾花数据集上表现出了良好的分类效果，具有高准确率和直观性。然而，它也存在一些潜在的局限性，如过拟合的风险、剪枝策略的选择以及特征选取的敏感性。为了进一步提升决策树分类器的性能，可以考虑以下改进方向：一是通过调整树的深度和剪枝策略来优化模型结构，避免过拟合；二是采用特征选择方法，选取更具代表性的特征，减少冗余信息对模型的影响；三是结合其他机器学习算法，如随机森林或梯度提升树等集成学习方法，来进一步提升分类性能和稳定性。未来实验可以围绕这些改进方向展开，并评估它们对决策树分类器性能的具体提升效果。 | | | |

1. 当次实验结束后一周内按班组织上交实验报告。

2. 实验报告文件命名为：“学号+姓名”，格式为WORD文档。