决策树作业报告

221900180 田永铭

2024 年 3 月 21 日

目录

实验任务

自选数据集实现决策树算法，并进行验证集评估，利用好 sklearn 库。

· 决策树是一种基于树状结构的模型，用于在给定输入数据的情况下进行决策。它通过一系列的分裂操作将数据集划分成多个子集，每个子集对应于一个决策路径，最终每个叶子节点表示一个类别标签或者一个数值输出。在机器学习中，决策树通常用于分类和回归任务。

数据的分析与处理

Iris 数据集 Iris 数据集是常用的分类实验数据集，由 Fisher, 1936 收集整理。Iris 也称鸢尾花卉数据集，是一类多重变量分析的数据集。数据集包含 150 个数据样本，分为 3 类，每类 50 个数据，每个数据包含 4 个属性。可通过花萼长度，花萼宽度，花瓣长度，花瓣宽度 4 个属性预测鸢尾花卉属于（Setosa，Versicolour，Virginica）三个种类中的哪一类。

利用代码调用数据集并且将数据归一化成可处理的数组:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
iris=load_iris()
x=np.array(iris.data)
y=np.array(iris.target)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

决策树的设计原理和核心代码

决策树的设计原理是基于对数据集的反复分割，以构建一个树状结构，每个节点都代表一个特征属性的测试条件。以下是决策树的设计原理的一般步骤：

1. **特征选择：**首先，从训练数据中选择一个特征作为根节点，并根据该特征将数据集分割成多个子集。特征选择的目标是选择那些能够最好地区分不同类别的特征。

2. **节点分割：**接着，对每个子集重复进行分割操作，通过选取最佳的特征和相应的分割条件（如阈值），将数据集进一步划分成更小的子集。这个过程会持续递归地进行，直到达到停止条件。

3. **停止准则：**在节点分割的过程中，需要定义何时停止分割节点。常见的停止准则包括节点的最大深度、节点包含的最小样本数、节点的不纯度等。当满足停止准则时，停止节点的分割，将节点标记为叶子节点。

4. **剪枝：**在构建完整个决策树后，为了避免过拟合，可以对树进行剪枝操作，去除一些不必要的节点或子树。剪枝的目的是简化模型，使其更具泛化能力。

5. **预测：**当新的样本进入决策树时，根据其特征值沿着树的路径向下移动，并最终到达叶子节点。叶子节点的类别标签或数值输出即为模型的预测结果。

决策树的设计原理和核心代码

决策树的设计原理是基于对数据的反复划分，每次划分都旨在最大程度地减少数据集的不纯度，从而使得每个子集更加纯净。这种递归的划分过程最终形成了一个树状结构，每个节点都代表一个特征的测试条件，每个叶子节点都代表一个类别标签或数值输出。决策树的设计原理使得它在分类和回归问题中都能有效地进行建模。

代码实现:

```
from sklearn import tree #决策树导入
clf=tree.DecisionTreeClassifier(criterion='gini',splitter='best') #以基尼
    系数作为评判准则
clf=clf.fit(x_train,y_train) #训练
```

验证集合评估结果与分析

添加测试代码如下:

```
y_pred = clf.predict(x_test)
score = accuracy_score(y_test,y_pred)
print("准确率:␣",score)
```

运行 decision_tree.py，获得测试集上决策树表现结果如下:

```
准确率:  0.9333333333333333
```

图: 验证集准确率

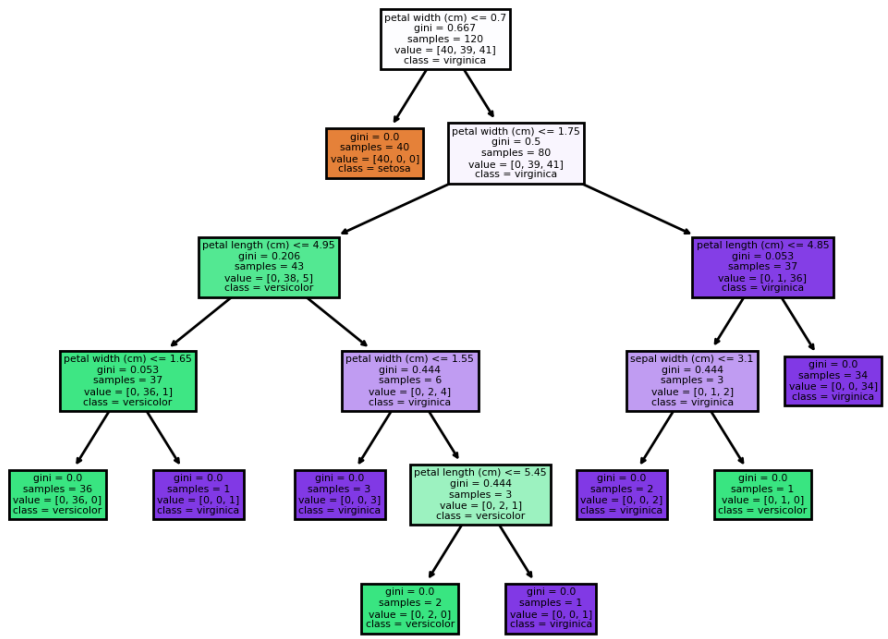
这个准确度已经达到了不错的效果。要想获得更加好的结果，有多方面的优化可以实现。比如说，防止决策树过拟合，采用其他评价损失和准确率的函数等等。接下来将实现。

可视化处理

添加可视化代码如下:

```
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
plt.figure(dpi=200)
tree.plot_tree(clf,feature_names=iris.feature_names,class_names=iris.
    target_names,filled=True)
plt.show()
```

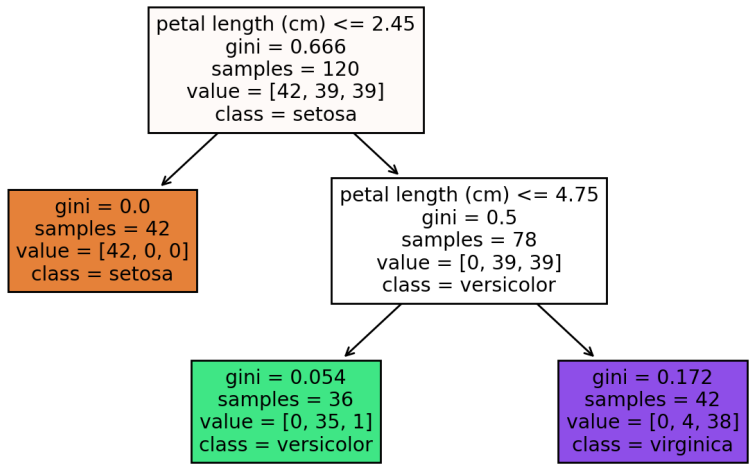
可以获得如下直观的决策树图:



加分 1: 剪枝

这段代码添加了剪枝参数 max_depth=2。这将限制决策树的最大深度为 2，从而避免过拟合。可以根据需要调整 max_depth 的值。

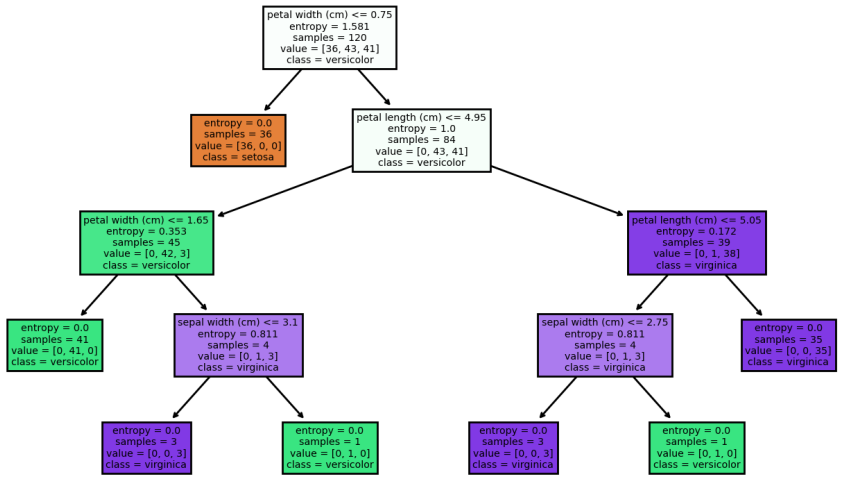
```
clf=tree.DecisionTreeClassifier(criterion='gini',splitter='best',
    max_depth=2)
```



可以观察到，剪枝后右下角少了分支。这是合理的，因为原本偏下方的一些类中只有极少数的样例，为此再分支不划算。同时，在这个样本上，准确率并没有大幅下降。

加分 2: 比较不同划分准则的决策树效果

对比采用信息增益 entropy，基尼系数 gini 两种不同准则下的效果：(只需修改代码中的 criterion)




```
准确率:  0.9666666666666667
```

可以发现，使用未剪枝的 entropy 比未剪枝的 gini 方法准确率略微高一些。

加分 3: 探索决策树在其他数据集上的表现

适当修改代码，考察决策树在 mnist 测试集上的表现 (绘图太大，略去):

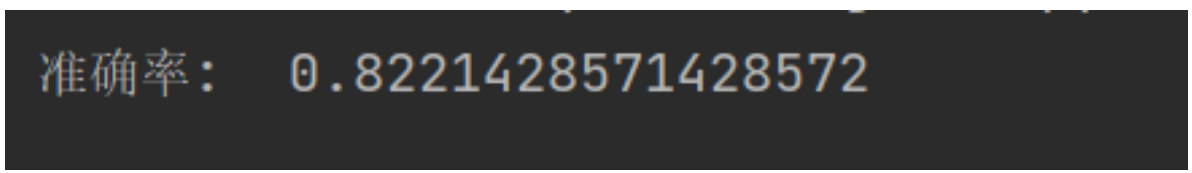
```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

# iris=load_iris()
# x=np.array(iris.data)
# y=np.array(iris.target)
mnist = fetch_openml('mnist_784', version=1, parser='auto')

x_train,x_test,y_train,y_test=train_test_split(mnist.data,mnist.target,
    test_size=0.2)

clf=tree.DecisionTreeClassifier(criterion='entropy',splitter='best',
    max_depth=8)
clf=clf.fit(x_train,y_train)

y_pred = clf.predict(x_test)
score = accuracy_score(y_test,y_pred)
print("准确率:␣",score)
```



加分 4: 取平均值看表现

注意到每次训练有波动，可以训练十次取平均值，这样更准确 (此为 iris 代码)：

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
iris = load_iris()
X = iris.data
y = iris.target
num_iterations = 10
accuracies = []
for i in range(num_iterations):
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=i)
clf = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth
    =3)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
score = accuracy_score(y_test, y_pred)
accuracies.append(score)

mean_accuracy = np.mean(accuracies)
print("平均准确率:␣", mean_accuracy)
```

```
平均准确率:  0.9466666666666667
```

总结

通过一些列的探索，我完成了很好的决策树模型，进行了很多拓展和优化，对决策树的理解更深了一步。