6.决策树算法

# 1 实验目的

1.理解sklearn.tree模块中决策树算法的实现原理；

2.了解sklearn.tree模块中极端决策树算法；

3.了解用sklearn提供的方法将决策树可视化；

4.了解如何使用sklearn.tree模块中各个常用算法完成机器学习任务。

# 2 实验要求

本次实验后，要求同学们能：

1.掌握决策树算法的实现原理；

2.学会使用Scikit-learn实现常用的决策树算法；

3.了解sklearn.tree模块中各常用算法的不同特点，了解各个超参数的含义。

# 3 实验内容

## 3.1 决策树算法简介

与SVM算法一样，决策树算法也是一个功能全面且强大的机器学习算法，能够很好地拟合复杂数据集。同时，作为一种白盒模型，决策树算法可解释性强，人们可以不仅可以用决策树进行决策，还可以学习到如何进行决策。

决策树也是随机森林的基本组成部分，随机森林是目前最强大的机器学习算法之一，会在第8章介绍。

下面将对Scikit-learn库中常用的决策树算法进行简单介绍。

## 3.2 决策树分类算法

### 3.2.1原理

用决策树进行分类主要步骤就是：①先将所有样本放在树的根节点，将样本的所有特征放在特征维度集。②从根节点开始，从特征维度集中选取一个特征，用这个特征对根节点进行划分，即，将根节点中的样本根据这个特征分配到它的子结点中。③之后再对子节点递归地进行相同的操作，直到把所有样本分配到叶结点中。

上述步骤中，主要需要解决两个问题：如何从特征维度集中选取划分父节点上样本的特征？以及如何判断是否已到达叶子节点？

决策树算法使用了“纯度”的概念来衡量划分的质量，可以解决这两个问题。“纯度”可以理解为是对单一类样本在子集内所占重的的度量。比如，经过某次划分后，某个子结点中属于A类的样本占50%，属于B类的样本占50%，那这个结点就是“不纯”的。如果子结点中属于A类的样本占100%，那这个结点就是“纯”的。

如果用某个特征对父结点进行划分后，其所有的子结点都是“不纯”的，甚至还没有划分之前的纯度高，那说明这次划分的效果很差，应该尝试选用别的特征进行划分。如果划分后某个结点纯度很高，达到一个可接受的值，就可以将这个子结点作为叶子结点。

纯度变化的衡量方式有很多，如信息增益、信息增益比、基尼指数等，因为信息增益要用到对数运算，效率较低，所以一般使用基尼指数生成决策树。下面简单介绍一下基尼指数和基尼指数构造决策树时用于优化的损失函数。

第i个节点上的基尼指数Gi计算公式如下，pi，k表示第i个节点上，实际属于k类的样本占比：

用于优化的成本函数为：

其中，k和tk分别表示划分父节点上样本时，选取的特征k和阈值tk。mleft / mright表示左/右子集的实例数量，Gleft / Gright表示左右子集的基尼不纯度。通过优化成本函数选取最优的特征和阈值，就可以对数据进行拟合。

但是，决策树极少对数据模型做出假设（比如线性模型会假设数据符合线性分布），自由度高，如果不对决策树加以限制，最终训练出的决策树很可能会严密地贴近训练集，虽然训练集上能达到很高的准确率，但是产生了过拟合，在测试集上表现不一定理想。

所以一般需要预先设置max\_depth（最大深度）、min\_samples\_split（可拆分结点的最小样本数）、min\_samples\_leaf（叶子结点的最小样本数）、max\_leaf\_nodes（叶子结点的最大个数）等超参数，对决策树的自由度进行限制，降低过拟合的风险。

### 3.2.2 实验

（1）数据预处理

用Scikit-learn自带的红酒数据作为实验数据，红酒数据集是一个经典且容易的多分类数据集，它包含3个分类，每个类别分别包含59、71、48个样本数据（总共178个样本），每个样本有13维的特征，分别是红酒中各个成分的含量信息。

将数据划分为训练集和测试集，决策树算法只涉及同一维特征之间的比较，所以不需要标准化或归一化。代码如下：

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier

from sklearn.datasets import load\_wine

from sklearn.model\_selection import train\_test\_split

# 用于可视化决策树

from sklearn import tree

from sklearn.tree import plot\_tree

from sklearn.tree import export\_text

# 加载数据集

wine\_data = load\_wine()

data = wine\_data.data

target = wine\_data.target

# 划分数据集

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

data,

target,

test\_size=0.2,

random\_state=12

)

（2）创建算法模型实例

用sklearn.tree模块的DecisionTreeClassifier类可以创建一个决策树分类算法实例，决策树算法的构造器可接受的参数众多，其中重要参数及含义如下：

**criterion：**数决策树算法很重要的一个参数，用来选择使用何种方法来度量树的划分质量，从而选择最好的划分来构造决策树。可以选择“**gini**”表示用基尼指数衡量划分质量，可以用于构建CART分类回归决策树；选择“**entropy**”表示用信息增益衡量划分质量，可以用于构建ID3决策树。“entropy”涉及对数运算，所以效率较低，一般情况下选择“gini”。

**splitter：**指定每个节点上选择划分条件的策略。“**best**”表示在特征的所有划分点中选择最优的一个，适合样本量不大的情形；“**random**”表示在随机的部分划分点中选择局部最优，可以构建随机决策树，适合样本量或特征数庞大的情形。

**max\_depth：**指定决策树的最大深度。如果取默认值None，则会将所有节点展开，直到所有的叶子节点都只有一个输出，或者包含少于min\_samples\_split个样本。

**min\_samples\_split：用于控制对节点的过度拆分。指定拆分节点处所需的最少样本数，当某个节点中的样本数小于这个指定值时，则不会再对该节点进一步划分，并把该节点当做叶子节点。min\_samples\_split可取int类型，代表样本个数；也可取float类型，代表占总样本数的比例。**

**min\_samples\_leaf： 用于剪枝。若经过父节点拆分得到的两个新的子节点样本数都低于min\_samples\_leaf，则将它们剪枝处理，令它们的父节点作为叶子节点。该值的设定可以防止模型过度拟合，让模型忽略少数（小于min\_samples\_leaf个）的异常点，从而让模型泛化能力更强。**

**max\_features：寻找最佳划分时要考虑的特征数量。取值为int类型时，表示在每个拆分节点考虑个特征；取值为float类型时，表示每次划分考虑个特征；取值为“auto”和“sqrt”时，表示每次划分考虑个特征；取值为“log2”时，表示每次划分考虑个特征；取值为“None”时，表示每次划分考虑全部特征。**

**max\_leaf\_nodes：限制最终产生的叶子节点数，防止过拟合。**

**min\_impurity\_decrease：设置节点划分的不纯度减少量最小值。如果划分前后，不纯度的减少量（纯度的增加量）小于min\_impurity\_decrease这个阈值，则取消本次划分。不纯度的减少量由基尼系数、信息增益、信息增益比等计算。**

**class\_weight：类别权重，只适用于分类树。可以指定样本各类别的的权重，主要是为了防止训练集某些类别的样本过多，导致训练的决策树过于偏向这些类别。可以以字典或字典列表的形式自定义；也可以选择“banlance”自动根据根据标签值，将权值调整为类别频率的反比（即，样本量少的类别有更高的权值）；默认值为“None”表示不设置权值。**

**本实验创建一个以基尼系数为划分质量评价指标的**CART决策树。由于红酒数据集样本数、特征数较小，所以其他参数设为默认值就可以得到良好的结果。创建决策树实例代码如下：

**model = DecisionTreeClassifier(criterion="gini")**

（3）训练并测试

除了predict\_log\_proba、predict\_proba、fit、predict、get\_params、get\_params、score等常用方法等常用方法，决策树算法实例还提供以下主要算法：

**decision\_path(X)：**返回决策路径。

**get\_depth()：**返回决策树深度。

**get\_n\_leaves()：**返回决策树的叶子节点个数。

用训练样本对决策树算法模型进行训练，检验模型的预测结果、在测试集上的评分。调用get\_depth()、get\_n\_leaves()查看训练出的决策树的形状信息。

# 训练

model.fit(X\_train, Y\_train)

# 评分

print("\n纯度的衡量方法为" + model.criterion + "时")

print("训练集得分为：%.6f" % model.score(X\_train, Y\_train))

print("测试集得分为：%.6f" % model.score(X\_test, Y\_test))

# 测试

print("由测试样本得到的预测值")

print(model.predict(X\_test))

print("实际值")

print(Y\_test)

print("决策树深度：", model.get\_depth())

print("决策树叶子节点个数：", model.get\_n\_leaves())

（4）查看训练后的参数

可以用parameter\_（参数名加下划线）的形式直接访问训练得到的以下参数：

**classes\_：分类器已知的分类标签。**

**feature\_importances\_：查看各个特征的重要程度。**

**n\_classes\_：查看待分类的类别数。**

**tree\_：返回一个基础的树对象，可以用sklearn.tree中的export\_text()或export\_graphviz()方法将决策树可视化。**

**实验代码如下：**

# 查看参数

print("\nclasses\_\n", model.classes\_)

print("feature\_importances\_\n", model.feature\_importances\_)

print("n\_classes\_\n", model.n\_classes\_)

print("tree\_\n", model.tree\_)

（5）可视化决策树

**用sklearn.tree中的export\_text()方法，以文本的形式查看训练出的决策树，代码如下：**

tree\_text = export\_text(

model,

feature\_names=wine\_data["feature\_names"],

show\_weights=True

)

print(tree\_text)

用sklearn.tree中的plot\_tree()方法，可以很方便地绘制出决策树图像，可以用matplotlib.pyplot.figure()设置图像大小，代码如下：

plt.figure(figsize=(9, 9))

tree.plot\_tree(model)

plt.show()

### 3.2.3 实验结果

（1）决策树分类模型的测试和评价

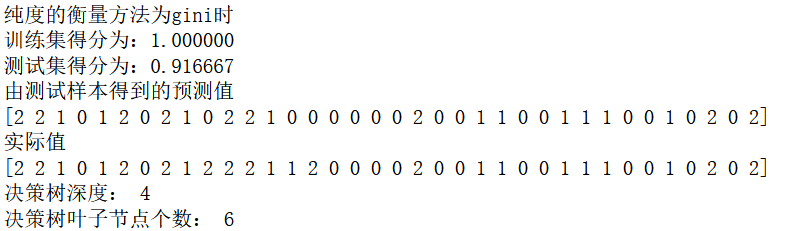


图1 决策树分类模型的测试和评价

（2）查看决策树参数

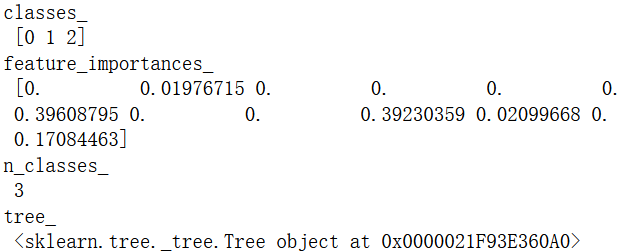


图2 查看决策树参数

（3）文本形式查看决策树信息

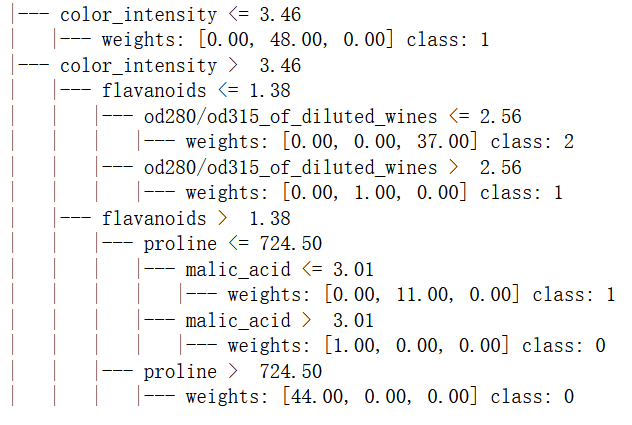


图3 文本形式查看决策树信息

（3）绘制决策树图像

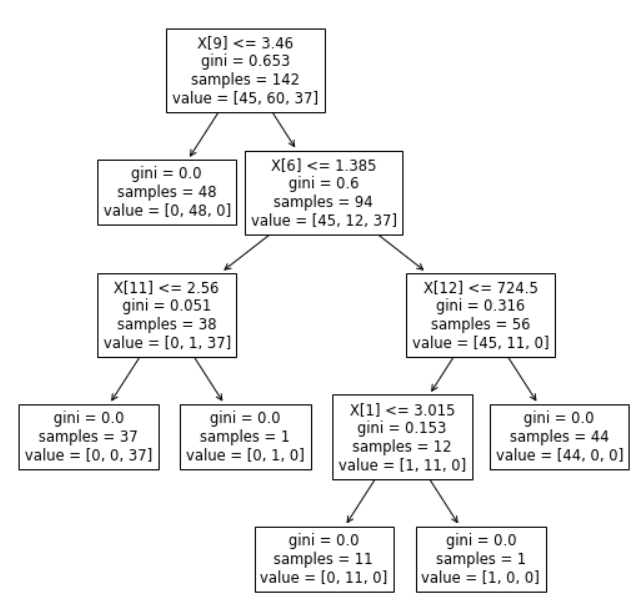


图4 决策树图像

## 3.3 决策树回归算法

### 3.3.1原理

决策树算法也可以用于解决回归问题。用决策树算法执行回归任务时，生成的决策树与分类决策树相似，主要差别在于每个结点不再是预测一个类别，而是预测一个值，每个区域的预测值等于这个区域内全部样本的平均值。

而衡量划分质量选择划分的方式也不再是最小化不纯度，而是最小化划分区域的均方误差MSE。用于选择划分方式的损失函数如下：

其中MSE是对应区域样本的均方误差，计算公式如下：

y(i)是结点中第i个样本的预测值

决策树在处理回归问题时也很容易过拟合，需要在训练前设置参数max\_depth（最大深度）、min\_samples\_split（可拆分结点最小样本数）、min\_samples\_leaf（叶子结点最小样本数）、max\_leaf\_nodes（叶子结点最大结点数）等超参数，对决策树的自由度进行限制，避免过拟合。下面实验就是以max\_depth为例，演示该参数对模型泛化能力的影响。

### 3.3.2 实验

决策树回归算法模型与决策树分类算法类似，模型实例的构造器接受同样的参数，只是有些参数的含义和默认值有些许不同。下面对决策树的深度进行实验，设置不同的**max\_depth值对决策树进行正则化，观察决策树深度对回归算法的影响。**

**（1）准备实验数据**

**准备100条含有服从高斯分布扰动的y = 2x + 10中的（x， y）作为实验数据，x作为特征，y作为标签，划分训练集和测试集。**

**import matplotlib.pyplot as plt**

**from sklearn.tree import DecisionTreeRegressor**

**from sklearn.linear\_model import LinearRegression**

**from sklearn.model\_selection import train\_test\_split**

**# 载入数据**

**data = np.array(range(100)).reshape(-1, 1)**

**target = 2 \* data + 10 + np.random.randn(100, 1) \* 20**

**X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(**

**data,**

**target,**

**test\_size=0.2,**

**random\_state=1,**

**)**

**Y\_test = np.sort(Y\_test, axis=0)**

**X\_test = np.sort(X\_test, axis=0)**

**（2）创建并训练模型**

**用sklearn.tree模块的DecisionTreeRegressor类构造决策树回归算法模型实例，其构造器的参数与决策树分类模型的构造器区别主要有以下两点：**

**criterion：决策树分类算法中，用“gini”或“**entropy**”衡量划分质量，寻找划分方式。而决策树回归算法中，用的是“squared\_error”均方误差，将方差减少量作为特征选择的准则，并使用每个终端节点的平均值来最小化L2损失； “friedman\_mse”在均方根误差基础上，使用Friedman改进评价分数来寻找潜在的划分方式；“absolute\_error”平均绝对误差，将平均绝对误差的减少量作为特征选择的准则，并用每个终端的中值来最小化L1损失；“poisson”用泊松偏差的减少来寻找划分方式。决策树回归算法默认使用“squared\_error”策略。**

**class\_weight：决策树回归算法中没有这个参数。**

**分别创建max\_depth为1、3、5的决策树回归模型和一个普通最小二乘线性回归模型进行比较。用训练数据进行训练。**

**model1 = DecisionTreeRegressor(max\_depth=1)**

**model2 = DecisionTreeRegressor(max\_depth=3)**

**model3 = DecisionTreeRegressor(max\_depth=5)**

**model4 = LinearRegression()**

**model1.fit(X\_train, Y\_train)**

**model2.fit(X\_train, Y\_train)**

**model3.fit(X\_train, Y\_train)**

**model4.fit(X\_train, Y\_train)**

**（3）评价与可视化**

**用matplotlib绘制各个模型的拟合结果图像，并分别打印出各个模型在训练集和测试集上的评分。**

**# 拟合图像**

**plt.figure(figsize=(10, 8))**

**plt.scatter(X\_test, Y\_test, s=40, c="black", label="data")**

**plt.plot(X\_test, model1.predict(X\_test), "co--", label="max\_depth=1", alpha=0.6)**

**plt.plot(X\_test, model2.predict(X\_test), "go--", label="max\_depth=3", alpha=0.6)**

**plt.plot(X\_test, model3.predict(X\_test), "ro--", label="max\_depth=5", alpha=0.6)**

**plt.plot(X\_test, model4.predict(X\_test), "bo--", label="liner regression", alpha=0.6)**

**plt.xlabel("data")**

**plt.ylabel("target")**

**plt.title("Decision Tree Regression")**

**plt.legend()**

**plt.show()**

**# 训练集上的评分**

**print("max\_depth=1 Train Score", model1.score(X\_train, Y\_train))**

**print("max\_depth=3 Train Score", model2.score(X\_train, Y\_train))**

**print("max\_depth=5 Train Score", model3.score(X\_train, Y\_train))**

**print("liner regression Train Score", model4.score(X\_train, Y\_train))**

**# 测试集上的评分**

**print("\nmax\_depth=1 Test Score", model1.score(X\_test, Y\_test))**

**print("max\_depth=3 Test Score", model2.score(X\_test, Y\_test))**

**print("max\_depth=5 Test Score", model3.score(X\_test, Y\_test))**

**print("liner regression Test Score", model4.score(X\_test, Y\_test))**

### 3.3.3 实验结果

（1）各个模型的拟合图像

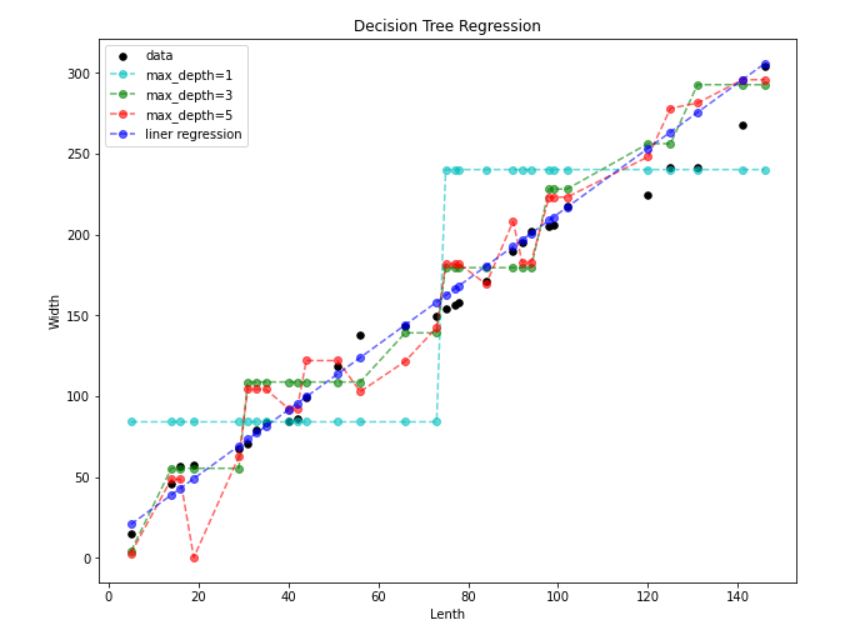


图5 各模型拟合图像

（2）各个模型的评分

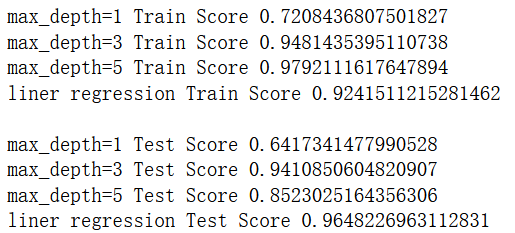


图6 各模型在训练集、测试集上评分

可以看到，决策树拟合带扰动的线性模型时中，决策树深度越高，训练集上的评分越高，但是在测试集上的评分却不一定更好。

决策树的深度可以增强回归决策树的对训练数据的拟合能力，但是过高的深度可能会导致泛化能力下降，产生过拟合现象。

## 3.4 极端随机决策树

### 3.4.1 原理

极端随机决策树（ExtraTreeClassifier与ExtraTreeRegressor）一般用于构建极端随机森林，是高度随机化的决策树，它不同于经典决策树，在寻找最佳划分时，随机选择max\_features个特征从中进行随机划分，并选取局部最佳的划分。当max\_features设置为1时，相当于构建了一个完全随机的决策树。

极端随机决策树（ExtraTreeClassifier与ExtraTreeRegressor）的参数构成与对应的经典决策树（DecisionTreeClassifier与DecisionTreeRegressor）一致，只是默认取值上有所差别。与经典决策树实例构造时的默认参数相比，构造极端随机决策树实例时，**splitter参数默认为“random”（经典决策树默认为“best”），max\_features参数默认为“sqrt”（经典决策树默认为“None”）**

### 3.4.2 实验

以分类决策树为例，将本章第一个实验改用极端随机决策树进行实验，只需更改一行代码，将创建算法模型时使用的构造函数由DecisionTreeClassifier()改为ExtraTreeClassifier()。

import matplotlib.pyplot as plt

from sklearn.tree import ExtraTreeClassifier

from sklearn.datasets import load\_wine

from sklearn.model\_selection import train\_test\_split

# 加载数据集

wine\_data = load\_wine()

data = wine\_data.data

target = wine\_data.target

# 划分数据集

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

data,

target,

test\_size=0.2,

random\_state=12

)

model = ExtraTreeClassifier(criterion="gini")

# 训练

model.fit(X\_train, Y\_train)

# 评分

print("\n纯度的衡量方法为" + model.criterion + "时")

print("训练集得分为：%.6f" % model.score(X\_train, Y\_train))

print("测试集得分为：%.6f" % model.score(X\_test, Y\_test))

# 测试

print("由测试样本得到的预测值")

print(model.predict(X\_test))

print("实际值")

print(Y\_test)

print("决策树深度：", model.get\_depth())

print("决策树叶子节点个数：", model.get\_n\_leaves())

# 查看参数

print("\nclasses\_\n", model.classes\_)

print("feature\_importances\_\n", model.feature\_importances\_)

print("n\_classes\_\n", model.n\_classes\_)

print("tree\_\n", model.tree\_)

### 3.4.3 实验结果

极端随机决策树每次运行结果不确定，有可能会得到更好的结果。某一次极端随机决策树的分类实验结果如下图：

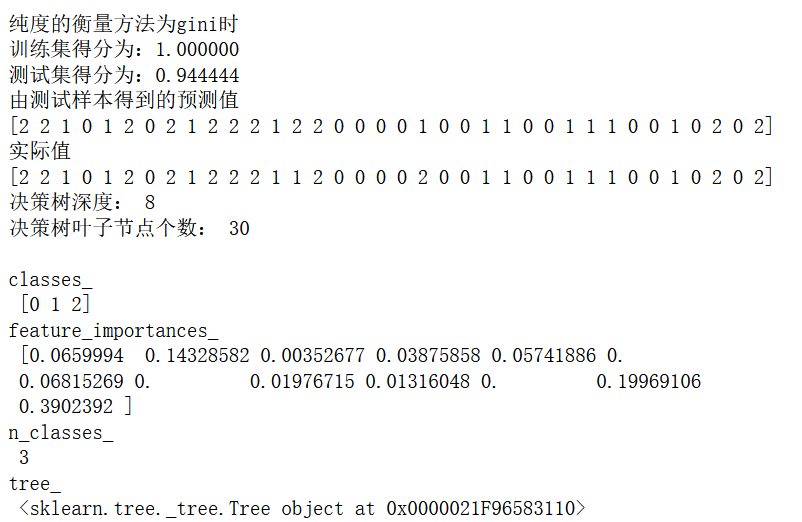


图7 极端随机决策树实验结果