机器学习应用实践（实验四）——决策树

一、实验目的

1、掌握 Scikit-Learn 中决策树算法的使用，能够使用分类树和回归树算法解决实际问题。

2、掌握决策树算法中的剪枝策略对其性能的影响。

3、掌握 CART 决策树算法的原理，并具备使用 python 实现决策树算法的编程能力。

4、掌握三种常用的类别数据编码方法。

二、实验准备

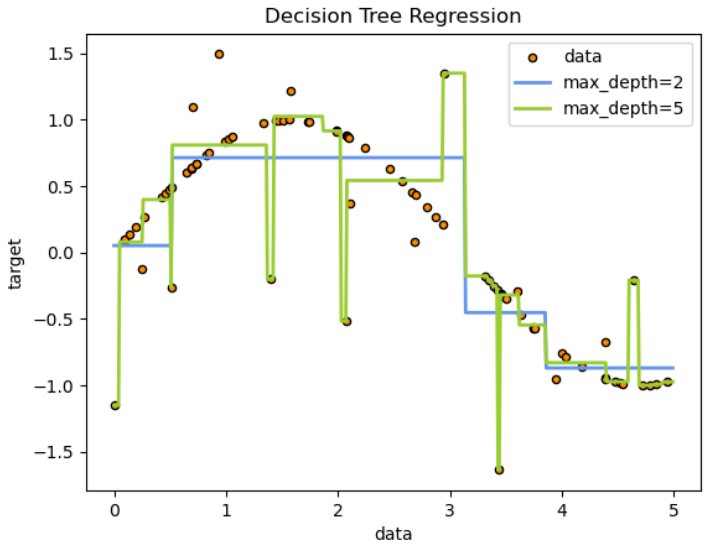
1、学习 Scikit-Learn 中决策树算法涉及到的相关知识。

2、回顾决策树算法和 CART 决策树算法的流程。

三、相关知识介绍

1、sklearn 官方文档对决策树的介绍。https://scikit-learn.org/stable/modules/tree.html

Decision Trees (DTs)决策树是一个非参数（不限制数据的结构和类型）的监督学习方法，可以用于分类和回归。决策树的目标是创建一个模型，通过学习从数据特征推断出的简单决策规则来预测目标变量的值。例如，在下图中，决策树用一系列 if-then-else 决策规则从数据中学习到一条近似的正弦曲线。树越深，决策规则和拟合的模型越复杂。



决策树具有如下优点：

（1）易于理解和解释，树可以被可视化。

（2）几乎不需要进行数据预处理。其他技术通常需要数据规范化，需要创建虚拟变量（哑变量），并删除空白值等。需要注意的是 sklearn 中的决策树不支持缺失值。

（3）使用树（如用来预测数据）的成本与用于训练树的数据点的数量成对数关系。

（4）决策树既能够处理数值变量，也能够处理分类变量。其他技术通常专门用于分析只有一种变量类型的数据集。

（5）能够处理多个标签变量（multi-output）的问题。

（6）决策树是一个白盒模型。如果在一个模型中某种情形是可以观测到的，那么这个条件很容易通过布尔逻辑进行解释。相反，在黑盒模型中（例如人工神经网络），结果解释起来很困难。

（7）能够使用统计检验验证模型，这使得解释模型具备可靠性。

（8）尽管它的假定在某种程度上可能有些违背生成数据的真实模型，决策树仍然表现良好。

决策树具有如下缺点：

（1）决策树可能创建过于复杂的树以至于不能很好的概括数据。这被称作是过拟合。为了避免这个问题，需要对决策树进行剪枝，设定一个叶子节点要求的最小样本量，或者设置树的最大深度。

（2）数据的微小变异可能导致决策树不稳定，进而生成一棵完全不同的决策树。结合集成算法使用决策树可以缓解这个问题。

（3）学习一颗最优决策树是一个 NP 问题，因此实际的决策树学习算法基于启发式算法（例如贪心算法），其在每个节点做出局部最优决策。这样的算法不能保证返回全局最优的决策树。通过有放回的随机抽取特征和样本来建立多棵树的集成算法可以使这个问题得到缓解。

（4）对于例如异或、奇偶校验或多路复用等问题，决策树不容易表达和学习。

（5）如果数据中某些类占主导，决策树学习器会学到一棵有偏差的树，因此我们推荐在拟合决策树之前应该平衡数据集。

决策树算法有：ID3，C4.5，C5.0 和 CART。

ID3（迭代二分法 3）由 Ross Quinlan 于 1986 年开发。该算法创建一个多路树，为每个节点（即以贪婪的方式）找到分类特征，该特征将为分类目标产生最大的信息增益。树会生长到最大尺寸，然后通常使用修剪步骤来提高树的泛化能力。

C4.5 是 ID3 的继承，它通过动态定义一个离散属性（基于数值变量），将连续属性值划分为一组离散的区间，消除了特征必须是离散值的限制。C4.5 将经过训练的树（即 ID3 算法的输出）转换为 if-then 规则集。然后评估每个规则的这些准确性，以确定应用它们的顺序。剪枝是通过删除规则的先决条件来完成的，前提是如果没有规则，规则的准确性就会提高。

C5.0 是 Quinlan 根据专有许可证发布的最新版本。与 C4.5 相比，它使用的内存更少，构建的规则集也更小，同时也更准确。

CART（分类和回归树）与 C4.5 非常相似，但它的不同之处在于它支持数字目标变量（回归），而且不计算规则集。CART 使用在每个节点产生最大信息增益的特征和阈值构建二叉树。sklearn 使用了 CART 算法的优化版本。

注意： 在 sklearn 的树模型介绍文档中，有一段关于 sklearn 的决策树不支持离散变量(categorical variables)建模的说明，其意为不支持按照类似 ID3 或 C4.5 的方式直接将离散变量按列来进行展开，而是根据 sklearn 中集成的 CART 树自身的建模规则,使得 sklearn 中的决策树实际上在处理特征时都是按照 C4.5 中连续变量的处理方式在进行处理，并非指的是带入离散变量就无法建模。

对于离散变量，可以进行 one-hot 编码处理，比如一个特征具有 3 个离散取值，每个离散取值都可将其转化为 one-hot 向量添加到属性中，最终变成 3 个特征，每个特征值只有 0 和 1。sklearn 中 one-hot 编码方法的相关使用见官网：

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html?highlight=one%20hot#sklearn.preprocessing.OneHotEncoder

2、sklearn 中决策树分类模型 DecisionTreeClassifier API 介绍

DecisionTreeClassifier 是 sklearn 中用来训练分类树的一个类对象，能够处理数据的多类别分类问题。DecisionTreeClassifier 接收两个数组作为输入，一个是数组 X，大小为(n_samples,n_features)，表示训练样本，另一个是整数数组 Y，大小为(n_samples,)，作为训练数据的类标签。

class sklearn.tree.DecisionTreeClassifier(criterion='gini', max_depth=None,random_state=None)

（1）基本参数：

●criterion：特征选择标准。有'gini'和'entropy'两个选项。默认是'gini'系数，也可以选择信息增益'entropy'。

●splitter：特征划分标准。有'best'和'random'两个选项。'best'是在特征的所有划分点（即随机选取的一部分特征）中找出最优的划分点。'random'是随机的在部分划分点中找局部最优的划分点。默认为'best'。

●random_state：用来设置分枝中的随机模式的参数。默认为 None。sklearn 在每次分枝时，不会使用全部特征，而是会随机选取一部分特征，从中选取指标最优（由 splitter 参数决定指标选取的方式）的作为分枝节点，每次生成的树也就不一样。在高维度时随机性会表现更明显，低维度时（比如鸢尾花数据集），随机性几乎不会体现。输入任意整数，会一直长出同一棵树，让模型稳定下来。

（2）剪枝参数：

在不加限制的情况下，一颗决策树会生长到衡量不纯度的指标最优，或者没有更多的特征可用为止，这样的决策树往往会过拟合，即在训练集上表现很好，在测试集上却表现不好。决策树模型对训练数据有了过于优秀的解释性，那么它找到的规则必然是包含了训练样本中的噪声的规则，这会使其对未知数据的拟合发生偏离，从而在测试集上表现不好。

为了让决策树有更好的泛化性，需要对决策树进行剪枝。剪枝策略对决策树影响巨大，正确的剪枝策略是优化决策树算法的核心。sklearn 提供了不同的剪枝策略，可通过如下参数进行设置：

●max_depth：树的深度最大深度（不包括根节点）。默认值为 None，表示节点将展开直到所有叶都是纯叶，或者直到所有叶都包含少于 min_samples_split 个 samples。这是用的最广泛的剪枝参数，在高维度低样本量时非常有效。决策树多生长一层，对样本量的需求会翻一倍，所以限制树的深度能够有效地限制过拟合。在集成算法中也非常实用。实际使用时，建议从 max_depth =3 开始尝试，依据拟合效果再决定是否增加最大深度。

●min_samples_split：内部节点再划分所需最小样本数。可以输入 int 和 float 数据，默认为 2。如果是 int，则取传入值本身作为最小样本数；如果是 float，则取 ceil(min_samples_split * 样本数量)

的值作为最小样本数，即向上取整。这个值限定了一个节点必须包含至少 min_samples_split 个样本，才被允许分枝，否则分枝就不会发生。

●min_samples_leaf：叶子节点最少样本数。可以输入 int 和 float 数据，默认为 1。如果是 int，则取传入值本身作为最小样本数；如果是 float（一般为百分比），则取 ceil(min_samples_leaf * 样本数量)的值作为最小样本数，即向上取整。这个值限制了叶子节点最少的样本数，如果某叶子节点数目小于 min_samples_leaf，则会和兄弟节点一起被剪枝，或者分枝会朝着满足每个叶子节点都包含 min_samples_leaf 个样本的方向去产生。

min_samples_leaf 一般搭配 max_depth 一起使用，在回归树中有神奇的效果，可以让模型变得更加平滑。min_samples_leaf 如果设置的太小，容易引起过拟合，设置的太大会阻止模型学习数据。一般建议从 min_samples_leaf=5 开始使用。对于类别不多的分类问题，min_samples_leaf=1 通常就是最佳选择。

●max_features：寻找最佳分割时要考虑的特征数量。可输入 int、float 或{‘auto’、’sqrt’、’log2’}。默认为 None，即 max_features 就是特征总数。’sqrt’则 max_features 是 sqrt(特征总数)。这个值限制了分枝时考虑的特征数，超过这个限制的特征都会被舍弃。这样做强行削减了数据的信息量，所以在不知道决策树中的各个特征的重要性的情况下，设定这个参数可能会导致模型学习不足。如果希望通过降维的方式防止过拟合，建议使用 PCA、ICA 或者特征选择模块中的降维算法。

●max_leaf_nodes：最大叶子节点数目。默认为 None，即不限制最大的叶子节点数。如果加了限制，算法会建立在最大叶子节点数内最优的决策树。

●min_impurity_decrease：最小不纯度下降。可输入 float，默认值为 0。当一个节点的加权不纯度下降大于 min_impurity_decrease 时，该节点就继续向下分裂。这个值限制信息增益（父节点的信息熵与子节点信息熵之差）的大小。信息增益越大，代表当前分枝的贡献越大，当需要进行大量计算得到的信息增益却较小时，没有必要再进行分枝，因此可以限制信息增益小于设定数值的分枝不会发生。

加权不纯度下降公式为：

```
N_t / N * (impurity - N_t_R / N_t * right_impurity
                    - N_t_L / N_t * left_impurity)
```

class_weight 和 min_weight_fraction_leaf 是完成样本标签平衡的参数。样本不平衡（样本偏斜）是指数据集中正负类样本数量不均，比如正类样本有 10000 个，负类样本只有 100 个，这样即使模型什么也不做，全把结果预测成负类，准确率也有 99%。因此我们要使用 class_weight 参数对样本标签进行一定的均衡，给少量的标签更多的权重，让模型更偏向少数类，向捕获少数类的方向建模。

●class_weight：以{class_label: weight}的形式表示与类别关联的权重。可输入 dict, list dict, 或’balanced’，默认为 None。如果取值为 None，则所有分类的权重为 1。

（a）对于多类别分类(Multiclass)问题，每个样本只有一个标签。当 class_weight 为“balanced”表示模式将自动调整权重，使之与输入数据中的各类占比成反比，公式为：n_samples/n_classes/np.bincount(y)，其中 n_samples 表示样本总数，n_classes 表示总类别数量，np.bincount(y)输出所有

类别的每个类别的样本数量，y 是所有样本的标签。n_samples/n_classes 表示样本均衡时每个类别的权重，再除以 np.bincount(y)表示根据每种类别中的样本数量对每个样本进行平均分配权重。

（b）对于多输出/多标签(Multioutput/Multilabel)问题，一个样本可以有多个标签，每个标签会有不少于 2 种的可能类别。例如，对一组水果图像的 “水果类型”和“颜色”进行分类。“水果类型”可能的类别有：“苹果”、“梨”和“橘子”。“颜色”可能的类别有：“绿色”、“红色”、“黄色”和“橙色”。每个样本都是水果的图像，都会输出两个标签。此时 class_weight 可以设置样本权重(sample_weight)，按照 y 的列的顺序提供一个字典列表，在其自己的字典中为每一列的每个类别定义权重。例如：对于四分类多标签问题，权重应为[{0:1,1:1},{0:1,1:5},{0:1,1:1},{0:1,1:1}]，而不是[{1:1},{2:5},{3:1},{4:1}]。

有了权重之后，样本量就不再是单纯地记录数目，而是受输入的权重影响了，因此这时候剪枝，就需要搭配 min_weight_fraction_leaf 这个基于权重的剪枝参数来使用。另请注意，基于权重的剪枝参数(例如 min_weight_fraction_leaf)将比不知道样本权重的参数(比如 min_samples_leaf)更少偏向主导类。如果样本是加权的，则使用基于权重的预修剪标准更容易优化树结构。

●min_weight_fraction_leaf：在叶子节点处（所有输入样本）的权重总和中的最小加权分数。可输入 float，默认为 0。如果未提供 sample_weight，则样本的权重相等。

●ccp_alpha：结构风险权重。ccp 是复杂度剪枝（Cost-Complexity Pruning）的简称，这是一个在 sklearn 的 0.22 版本中才加入的参数，这也是唯一一个为实现 CART 原生原理中的剪枝过程所设置的参数。此处首先需要知道的是在 sklearn 中并不一定要通过该方法进行剪枝，因此该参数其实也并不是一个必选参数。其次，带有 ccp 项的剪枝也被称为最小复杂度剪枝，其原理是在决策树的损失函数上加上一个结构风险项，类似于正则化项在线性方程的损失函数中作用。 我们可以设 T 为某决策树，R(T)为决策树在训练集上整体不纯度，即代表模型的经验风险，令 $\alpha|\widetilde{T}|$表示模型结构风险，其中 α 为参数，$|\widetilde{T}|$为树的叶节点数量，则我们可以修改模型损失函数如下：

$$R_\alpha(T)=R(T)+\alpha\left|\widetilde{T}\right|$$

α 参数是风险结构项的系数，其取值越大、对模型的结构风险惩罚力度就越大、模型结构就越简单、过拟合就能够被更好的抑制，反之亦反。

（3）重要属性：

●classes_：输出所有标签。输出一个数组(array)或者一个数组的列表(list)，结构为标签的数目(n_classes)。

●feature_importances_：输出一个数组，结构为特征的数目(n_features)，数据对应每个特征的重要性，一般是这个特征在多次分枝中产生的信息增益的综合，也被称为“基尼重要性”(Gini importance)，值越大表示特征越重要。在 sklearn 当中，feature_importances_是特征对 criterion 下降量的总贡献量。

●max_features_：输出整数。参数 max_features 的推断值。

●n_classes_：输出整数或列表。标签类别的数量（单一输出问题）。

●n_features_：在训练模型(fit)时使用的特征个数。

●n_outputs_：在训练模型(fit)时输出的结果的个数。

●tree_：树结构属性。决策树分类器的一个属性，能够以多个数组的形式存储整个二叉树结构信息，每个数组中的 i 号元素对应着节点 i 的信息，数组中 0 号元素对应树的根节点信息。可访问的信息包括节点数、节点总数和最大深度等，详见表 1。其中有些数组是只适用于叶子节点或者分裂节点的，比如用于分裂的特征和阈值就只适用于分裂节点，对于叶子节点分裂的特征和阈值就可以是任意值。

表 1 决策树结构属性 tree_中详细参数列表

参数	解释
tree_. node_count	训练得到的决策树的总节点数（包括内部节点+叶子节点）
tree_. capacity	树结构的数组大小
tree_. max_depth	树的最大深度
tree_.n_leaves	树的叶子节点的数量
tree_. children_left [i]	左分支数组，索引 i 对应的元素为节点 i 的左分支节点编号，若没有左分支，则对应元素为-1
tree_. children_right [i]	左分支数组，索引 i 对应的元素为节点 i 的右分支节点编号，若没有右分支，则对应元素为-1
tree_.feature[i]	分裂特征数组，索引 i 对应的元素为节点 i 的分裂特征，若为叶子节点，则对应元素为-2
tree_.threshold[i]	分裂阈值数组，索引 i 对应的元素为节点 i 的分裂阈值，若为叶子节点，则对应元素为-2
tree_.impurity[i]	不纯度数组，索引 i 对应的元素为节点 i 的不纯度，若为叶子节点，则不纯度为 0
tree_. n_node_samples[i]	每个节点的样本量数组，索引 i 对应的元素为节点 i 的样本量
tree_.value[i]	统计每个节点处的样本类别数量，索引 i 对应的元素为节点 i 的每个类别样本的数量

参考网址：

https://scikit-learn.org/stable/auto_examples/tree/plot_unveil_tree_structure.html#sphx-glr-auto-examples-tree-plot-unveil-tree-structure-py。

（4）重要方法：

方法	说明
apply(X[, check_input])	返回每个叶子节点上被预测样本的索引。
cost_complexity_pruning_path(X, y[, …])	在最小化成本复杂性修剪期间计算修剪路径。
decision_path(X[, check_input])	返回决策树的决策路径。
fit(X, y[, sample_weight, check_input, …])	根据训练集（X，y）建立决策树分类器。
get_depth()	返回决策树的深度。

方法	说明
get_n_leaves()	返回决策树的叶子数。
get_params([deep])	获取此估算器的参数。
predict(X[, check_input])	预测 X 的类别或回归值。
predict_log_proba(X)	预测输入样本 X 的类对数概率。
predict_proba(X[, check_input])	预测输入样本 X 的类别概率。
score(X, y[, sample_weight])	返回给定测试数据和标签上的平均准确度。
set_params(**params)	设置此估算器的参数。

DecisionTreeClassifier 使用的程序示例：

```
>>> from sklearn import tree
>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, Y)
```

拟合之后，模型就可以用来预测样本的类别，预测时只需要输入测试集的特征：

```
>>> clf.predict([[2., 2.]])
array([1])
```

也可以预测每个类别的概率，这个概率是同一个叶子节点中同类训练样本的比例。

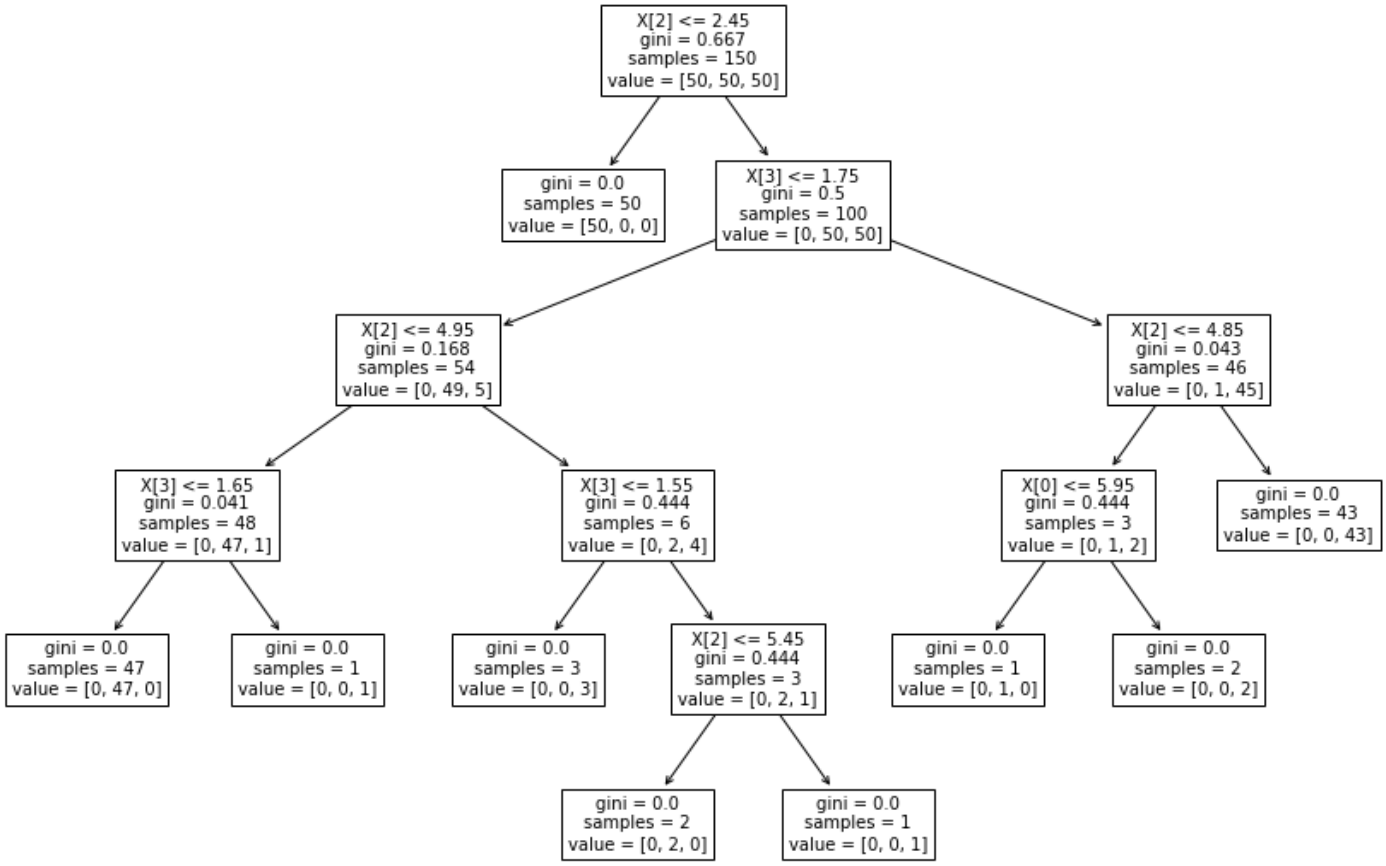
```
>>> clf.predict_proba([[2., 2.]])
array([[0., 1.]])
```

DecisionTreeClassifier 能够处理二分类（标签为[-1,1]）和多分类问题（标签为[0,1,……,K-1]）。使用 iris 数据集，我们可以以如下的方式构建一棵树：

```
>>> from sklearn.datasets import load_iris
>>> from sklearn import tree
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, y)
```

一旦训练完毕，可以使用 plot_tree 函数来画出树形图（使用 matplotlib 库），下面为 jupyter 代码示例，若用 pycharm，可用“plt.show()”显示决策树。

```
>>> tree.plot_tree(clf)
[...]
```



如果树形图不清晰，可通过设置字体大小将文字调大，以及调整画布大小。程序示例：

```
plt.figure(figsize=(15,10))
tree.plot_tree(clf,fontsize=10)
```

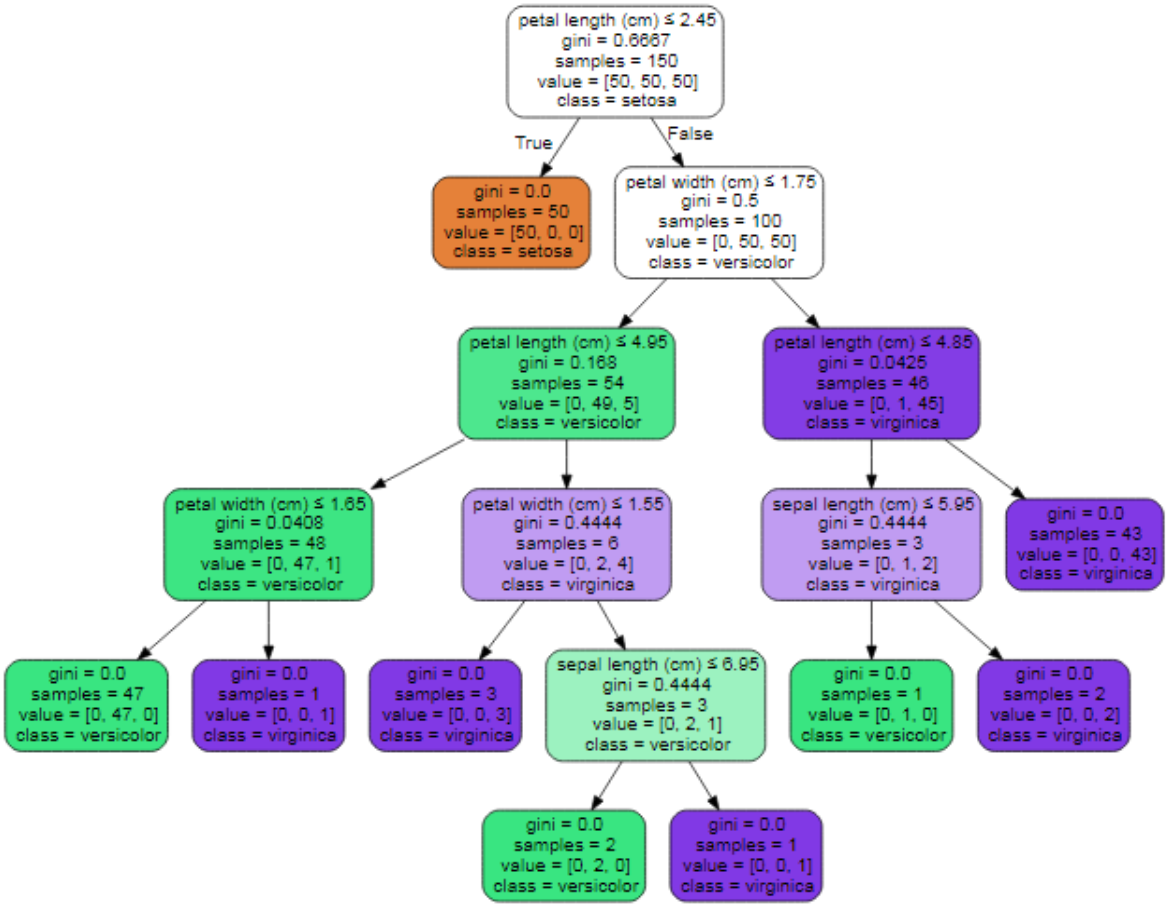
也可以使用 export_graphviz 工具以 Graphviz 格式导出一棵树。如果你使用了 conda 包管理工具，可以使用 conda install python-graphviz 命令安装 graphviz 二进制文件和 python 包。或者可以从 graphviz 项目主页下载 graphviz 二进制文件，然后从命令窗口中使用 pip install graphviz 命令来安装。

下面的例子导出了上面训练的树，结果保存在 iris.pdf 文件中。

```
>>> import graphviz
>>> dot_data = tree.export_graphviz(clf, out_file=None)
>>> graph = graphviz.Source(dot_data)
>>> graph.render("iris")
```

export_graphviz 导出器也支持许多美化功能的参数，包括根据类别或者回归值给节点绘制颜色，下面为 jupyter 代码示例，若用 pycharm，可用“graph.view()”显示决策树。参数设置中“filled=True”表示节点填充颜色(不是白色)，“rounded=True”表示节点绘图形状为圆角矩形。可以看出节点不纯度越低，颜色越深。

```
>>> dot_data = tree.export_graphviz(clf, out_file=None,
...                      feature_names=iris.feature_names,
...                      class_names=iris.target_names,
...                      filled=True, rounded=True,
...                      special_characters=True)
>>> graph = graphviz.Source(dot_data)
>>> graph
```



也可以使用 export_text 函数以文本的形式导出树，这种方法不需要安装外部库：

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.tree import DecisionTreeClassifier
>>> from sklearn.tree import export_text
>>> iris = load_iris()
>>> decision_tree = DecisionTreeClassifier(random_state=0, max_depth=2)
>>> decision_tree = decision_tree.fit(iris.data, iris.target)
>>> r = export_text(decision_tree, feature_names=iris['feature_names'])
>>> print(r)
|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) >  0.80
|   |--- petal width (cm) <= 1.75
|   |   |--- class: 1
|   |--- petal width (cm) >  1.75
|   |   |--- class: 2
```

决策树还可以用到回归问题中，需要使用 DecisionTreeRegressor 类。与分类问题中的设置相同，fit 方法接收数组 X 和 y 作为参数，在这种情况下，y 应该是浮点数而非整数。

class sklearn.tree.DecisionTree Regressor (criterion=’ squared_error’, max_depth=None, random_state=None)

DecisionTree Regressor 与 DecisionTreeClassifier 的不同参数：

●criterion：特征选择标准。回归树的特征选择标准包括’squared_error’, ’friedman_mse’, ’poisson’, ’absolute_error’}，默认为’squared_error’。

```
>>> from sklearn import tree
>>> X = [[0, 0], [2, 2]]
>>> y = [0.5, 2.5]
>>> clf = tree.DecisionTreeRegressor()
>>> clf = clf.fit(X, y)
>>> clf.predict([[1, 1]])
array([0.5])
```

3、CART 决策树算法

（1）CART 分类树

输入	训练集 $D=\{(x_1,y_1),(x_2,y_2),\dots,(x_m,y_m)\}$ 特征集 $F=\{f_l \mid l=1,\dots,L\}$
	函数TreeGenerate (D,F)
步骤	
1	生成节点node，并赋予对应的训练集D
2	**if** $\lvert D\rvert \le T_N$ **OR** $\mathrm{Gini}(D) \le T_G$ **then** //终止条件：T_N 为样本数量阈值，T_G 为基尼指数阈值
3	将node记为叶子节点，将类别标记为D中样本数最多的类；**return**
4	**end if**
5	**for** f_l **in** F　//遍历所有特征
6	if f_l 是离散特征
7	for f_l 的每一个取值f_l^k do　//遍历当前离散特征的所有取值
8	根据$f_l=$ or $\neq f_l^k$，将D分为左右两个子集D_{left}和D_{right}，记录加权平均基尼指数
9	else if f_l 是连续特征
10	for f_l 的每一个分裂阈值 t_l^k do　//遍历当前连续特征下的所有分裂阈值
11	根据$f_l >$ or $< t_l^k$？，将D分为左右两个子集D_{left}和D_{right}，记录加权平均基尼指数
13	根据最小加权平均基尼指数从F中选择最优划分属性f_l，及其对应的最优分裂方案给出的D_{left}和D_{right}
14	$D=D_{\text{left}}$，以TreeGenerate (D,F)向下继续生长
15	$D=D_{\text{right}}$，以TreeGenerate (D,F)向下继续生长
16	生成以node为根节点的决策树T'，对T'进行后剪枝
输出	得到最终的决策树T

（2）CART 回归树

输入	训练集 $D=\{(x_1,y_1),(x_2,y_2),\dots,(x_m,y_m)\}$ 特征集 $F=\{f_l \mid l=1,\dots,L\}$
	函数TreeGenerate (D,F)
步骤	
1	生成节点node，并赋予对应的训练集D
2	if $\lvert D\rvert \le T_N$ OR $V(Y) \le T_V$ then //终止条件：T_N 为样本数量阈值，T_V 为回归标签方差的阈值
3	将node标记为叶子节点，将回归标签集合Y的平均值$\bar{y}$记为node的回归值；return
4	end if
5	for f_l in F
6	if f_l 是离散特征
7	for f_l 的每一个取值f_l^k do
8	根据$f_l=$ or $\neq f_l^k$，将D分为左右两个子集D_{left}和D_{right}，记录加权平均误差
9	else if f_l 是连续特征
10	for f_l 的每一个分裂阈值 t_l^k do
11	根据$f_l >$ or $< t_l^k$？，将D分为左右两个子集D_{left}和D_{right}，记录加权平均误差
13	根据最小加权平均误差从F中选择最优划分属性f_l，及其对应的最优分裂方案给出的D_{left}和D_{right}
14	$D=D_{\text{left}}$，以TreeGenerate (D,F)向下继续生长
15	$D=D_{\text{right}}$，以TreeGenerate (D,F)向下继续生长
16	生成以node为根节点的决策树T'，对T'进行后剪枝
输出	得到最终的决策树T

4、数据预处理——类别数据编码

我们常会遇到数据集不是连续值，而是类别数据的形式。例如，一个人可能有[“男”、“女”]、[“来自欧洲”、“来自美国”、“来自亚洲”]、[“使用 Firefox”、“使用 Chrome”、“使用 Safari”、“使用 Internet Explorer”]。为了便于计算，可以将这些特征编码为整数，例如[“男性”、“来自

美国”、“使用 Internet Explorer”]，可以表示为[0、1、3]的形式，而[“女性”、“来自亚洲”、“使用 Chrome”]可以表示为[1、2、1]。

sklearn 提供了 LabelEncoder、OrdinalEncoder、OneHotEncoder 等类用于进行数据编码。其中，LabelEncoder 用于标签编码，OrdinalEncoder、OneHotEncoder 用于特征数据编码。

（1）LabelEncoder ：多类别标签编码

sklearn 中提供了 LabelEncoder 用来对多类别标签进行编码，将标签值转换成 0 到 n_classes-1 的整数。可以对本文编码，也可以对数字编码。

官网地址：https://scikit-learn.org/stable/modules/preprocessing_targets.html。

其常用方法有：

fit(y)：y 为多类别的标签值，训练标签编码器。

transform(y) ：使用训练好的编码器进行标签编码变换，并返回编码后的标签值。

fit_transform(y)：相当于先进行 fit 再进行 transform 转换，并返回编码后的标签值。

inverse_transform(y)：将标签编码进行反变换，并返回变换后的值。

需要注意的是，LabelEncoder 类在使用上述四种常用方法时只接收一维数组的形式。

对数字进行编码的程序示例：

```
>>> from sklearn import preprocessing
>>> le = preprocessing.LabelEncoder()
>>> le.fit([1, 2, 2, 6])
LabelEncoder()
>>> le.classes_
array([1, 2, 6])
>>> le.transform([1, 1, 2, 6])
array([0, 0, 1, 2])
>>> le.inverse_transform([0, 0, 1, 2])
array([1, 1, 2, 6])
```

对文本进行编码的程序示例：

```
>>> le = preprocessing.LabelEncoder()
>>> le.fit(["paris", "paris", "tokyo", "amsterdam"])
LabelEncoder()
>>> list(le.classes_)
['amsterdam', 'paris', 'tokyo']
>>> le.transform(["tokyo", "tokyo", "paris"])
array([2, 2, 1]...)
>>> list(le.inverse_transform([2, 2, 1]))
['tokyo', 'tokyo', 'paris']
```

（2）OrdinalEncoder：多类别特征编码（有序类别特征）

OrdinalEncoder 和 OneHotEncoder 类可用于类别特征编码。官网地址：https://scikit-learn.org/stable/modules/preprocessing.html#encoding-categorical-features。

其常用方法有：

fit(X)：X 为待编码的特征，训练标签编码器。

transform(X) ：按照求解器找到的最优参数进行转换，并返回编码后的特征值。

fit_transform(X)：相当于先进行 fit 再进行 transform 转换，并返回编码后的特征值。

inverse_transform(X)：将特征编码进行反变换，得到原始值。

需要注意的是，OrdinalEncoder 和 OneHotEncoder 类在使用上述四种常用方法时只接收二维数组的形式。因此，如果使用 pandas 进行数据处理，需要进行转换成数组类型。

OrdinalEncoder 将类别特征转换为整数编码的形式，每个特征中的一个类别用一个整数表示。每个类别特征转换为一个新的整数特征（整数的取值：从 0 到 n_categories - 1）。适用于本身有顺序的类别特征（比如年龄分层、学历等）。程序示例如下：

```
>>> enc = preprocessing.OrdinalEncoder()
>>> X = [['male', 'from US', 'uses Safari'], ['female', 'from Europe', 'uses Firefox']]
>>> enc.fit(X)
OrdinalEncoder()
>>> enc.transform([['female', 'from US', 'uses Safari']])
array([[0., 1., 1.]])
```

如果数据中有缺失值，OrdinalEncoder 会将缺失值用 np.nan 代替。程序示例如下：

```
>>> enc = preprocessing.OrdinalEncoder()
>>> X = [['male'], ['female'], [np.nan], ['female']]
>>> enc.fit_transform(X)
array([[ 1.],
       [ 0.],
       [nan],
       [ 0.]])
```

（3）OneHotEncoder：独热编码（无序类别特征）

OneHotEncoder（独热编码）是将每一个类可能取值的特征变换为二进制特征向量，每一类的特征向量只有一个位置是 1，其余位置都是 0。适用于无序的类别特征。onehot 编码后的特征无需无量纲化处理。onehot 编码也有缺陷，当类别的数量很多时，特征空间会变得非常大，计算复杂度会变高，成本也会变大。

重要参数：

●sparse：默认为 True，如果设置为 True 将返回稀疏矩阵，否则将返回一个数组。

●categories：默认为’auto’，可以通过参数 categories 指定需要保留的取值（指定后可以通过 ohe.categories_查看）。可选为 list，categories[i]保存第 i 列中预期的类别。

```
>>> X = [['male', 'Safari'],
...      ['female', None],
...      [np.nan, 'Firefox']]
>>> enc = preprocessing.OneHotEncoder(handle_unknown='error').fit(X)
>>> enc.categories_
[array(['female', 'male', nan], dtype=object),
 array(['Firefox', 'Safari', None], dtype=object)]
>>> enc.transform(X).toarray()
array([[0., 1., 0., 0., 1., 0.],
       [1., 0., 0., 0., 0., 1.],
       [0., 0., 1., 1., 0., 0.]])
```

程序中 enc.transform(X)将返回稀疏矩阵形式的 onehot 编码，使用.toarray()可将其转换成数组形式。也可以设置参数 sparse=False，则 enc.transform(X)将直接返回数组形式，无需使用.toarray()。除此之外，pandas 中的 get_dummies()函数也具有与 OneHotEncoder 相同的功能，感兴趣的同学可以自行尝试。

四、实验内容

1、题目一：采用 scikit-learn 中的 DecisionTreeClassifier 决策树对葡萄酒数据集进行预测。

具体要求：

（1）导入数据集：葡萄酒数据集是 sklearn 中自带的数据集，API 使用参考网址：https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html#sklearn.datasets.load_wine。点击“User Guide”可以查看数据集描述信息。

sklearn.datasets.load_wine

sklearn.datasets.**load_wine**(*, *return_X_y=False*, *as_frame=False*)

Load and return the wine dataset (classification).

New in version 0.18.

The wine dataset is a classic and very easy multi-class classification dataset.

Classes	3
Samples per class	[59,71,48]
Samples total	178
Dimensionality	13
Features	real, positive

The copy of UCI ML Wine Data Set dataset is downloaded and modified to fit standard format from:
https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

Read more in the User Guide.

通过查看数据量和维度、特征类型（离散 or 连续）、特征名、标签名、标签分布情况、数据集的描述等信息了解数据集。

（2）模型建立：使用全部特征建立决策树多分类模型（树模型参数可按默认设置）。

（3）输出：特征重要程度、分类准确率、绘制树形图。

【讨论一】模型参数对模型性能有何影响？

（1）不同特征选择标准（criterion ='gini'或者'entropy'）对模型性能是否有影响？

（2）不同特征划分标准（splitter='best'或者'random'）对模型性能是否有影响？

（3）尝试修改 max_depth、min_samples_leaf、min_samples_split 等参数，通过树形图分析参数的作用。

【讨论二】如何确定最优的剪枝参数？

找到合适的超参数，展示调整后的模型效果。可使用学习曲线进行超参数选取，其优点是可以看到超参数对模型性能影响的趋势。

如果需要确定的超参数比较多，且超参数之间相互影响时，可尝试使用 GridSearchCV 选择模型最优的超参数。

2、题目二（选做）：使用 scikit-learn 中的 DecisionTreeClassifier 决策树对 kddcup99 数据集进行预测。

提示：

（1）kddcup99 数据集是 KDD 竞赛在 1999 年举行时采用的数据集，是网络入侵检测领域的真实数据，具有 41 个特征，类别数量有 23 种(1 种正常网络、22 种网络入侵类型)，约五百万数据量。由于数据量很大，可以选择获取总数据量的 10%。

sklearn.datasets.fetch_kddcup99(, subset=None, data_home=None, shuffle=False, random_state=None, percent10=True, download_if_missing=True, return_X_y=False, as_frame=False)*

sklearn 官网地址：

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_kddcup99.html#sklearn.datasets.fetch_kddcup99。

Kaggle 官网数据源地址：http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html。

（2）树模型只能处理数值型数据，不能处理字母或文本数据。kddcup99 数据集中的第 2.3.4 列特征为文本信息，需要重新编码。类别标签也为文本数据，也需要编码操作。试选择适合的编码方式，对数据进行编码之后，再进行建模。展示编码结果。

（3）选择适合的评价指标，尝试调参，建立效果最好的一个模型。

3、题目三：使用 numpy 编写的 CART 分类/回归树（选择一种即可）算法，并对 iris 数据集/california 数据集进行预测。

具体内容：

（1）导入数据集。

（2）划分数据（分成训练集和数据集）

（3）训练模型（参考程序模板：cart_numpy_template.py）

（4）输出树模型。

（5）进行预测，评估模型性能。

拓展内容（选做）：

（1）尝试加入 T_N 样本数量阈值和 T_G 基尼指数阈值作为终止条件。

（2）尝试对离散特征进行分枝。

【讨论四】树递归分枝的终止条件是什么？展示对应的代码。请结合代码简述树的递归分枝的过程。