菜菜的scikit-klearn课堂第一期

sklearn入门 & 决策树在sklearn中的实现

小伙伴们晚上好~o(￣▽￣)ブ

我是菜菜，这里是我的sklearn课堂，今晚的直播内容是sklearn入门 + 决策树在sklearn中的实现和调参~

请扫码进群领取课件和代码源文件，**扫描二维码后回复"K"就可以进群哦~**

我的开发环境是**Jupyter lab**，所用的库和版本大家参考：

Python 3.7.1（你的版本至少要3.4以上

Scikit-learn 0.20.0 （你的版本至少要0.19

Graphviz 0.8.4 (没有画不出决策树哦，安装代码conda install python-graphviz

Numpy 1.15.3, **Pandas** 0.23.4, **Matplotlib** 3.0.1, **SciPy** 1.1.0

菜菜的scikit-klearn课堂第一期
sklearn入门 & 决策树在sklearn中的实现
sklearn入门
决策树

sklearn入门

scikit-learn，又写作sklearn，是一个开源的基于python语言的机器学习工具包。它通过NumPy, SciPy和Matplotlib等python数值计算的库实现高效的算法应用，并且涵盖了几乎所有主流机器学习算法。

http://scikit-learn.org/stable/index.html

在工程应用中，用python手写代码来从头实现一个算法的可能性非常低，这样不仅耗时耗力，还不一定能够写出构架清晰，稳定性强的模型。更多情况下，是分析采集到的数据，根据数据特征选择适合的算法，在工具包中调用算法，调整算法的参数，获取需要的信息，从而实现算法效率和效果之间的平衡。而sklearn，正是这样一个可以帮助我们高效实现算法应用的工具包。

sklearn有一个完整而丰富的官网，里面讲解了基于sklearn对所有算法的实现和简单应用。然而，这个官网是全英文的，并且现在没有特别理想的中文接口，市面上也没有针对sklearn非常好的书。因此，这门课的目的就是由简向繁地向大家解析sklearn的全面应用，帮助大家了解不同的机器学习算法有哪些可调参数，有哪些可用接口，这些接口和参数对算法来说有什么含义，又会对算法的性能及准确性有什么影响。我们会讲解sklearn中对算法的说明，调参，属性，接口，以及实例应用。注意，本门课程的讲解不会涉及详细的算法原理，只会专注于算法在sklearn中的实现，如果希望详细了解算法的原理，建议阅读下面这本两本书：

决策树

1 概述

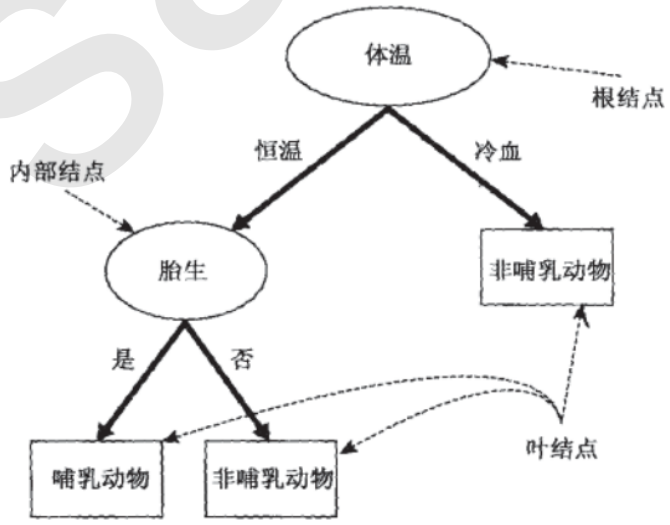
1.1 决策树是如何工作的

决策树（Decision Tree）是一种非参数的有监督学习方法，它能够从一系列有特征和标签的数据中总结出决策规则，并用树状图的结构来呈现这些规则，以解决分类和回归问题。决策树算法容易理解，适用各种数据，在解决各种问题时都有良好表现，尤其是以树模型为核心的各种集成算法，在各个行业和领域都有广泛的应用。

我们来简单了解一下决策树是如何工作的。决策树算法的本质是一种图结构，我们只需要问一系列问题就可以对数据进行分类了。比如说，来看看下面这组数据集，这是一系列已知物种以及所属类别的数据：

名字	体温	表皮覆盖	胎生	水生动物	飞行动物	有腿	冬眠	类标号
人类	恒温	毛发	是	否	否	是	否	哺乳类
鲑鱼	冷血	鳞片	否	是	否	否	否	鱼类
鲸	恒温	毛发	是	是	否	否	否	哺乳类
青蛙	冷血	无	否	半	否	是	是	两栖类
巨蜥	冷血	鳞片	否	否	否	是	否	爬行类
蝙蝠	恒温	毛发	是	否	是	是	是	哺乳类
鸽子	恒温	羽毛	否	否	是	是	否	鸟类
猫	恒温	软毛	是	否	否	是	否	哺乳类
豹纹鲨	冷血	鳞片	是	是	否	否	否	鱼类
海龟	冷血	鳞片	否	半	否	是	否	爬行类
企鹅	恒温	羽毛	否	半	否	是	否	鸟类
豪猪	恒温	刚毛	是	否	否	是	是	哺乳类
鳗	冷血	鳞片	否	是	否	否	否	鱼类
蠎蠎	冷血	无	否	半	否	是	是	两栖类

我们现在的目标是，将动物们分为哺乳类和非哺乳类。那根据已经收集到的数据，决策树算法为我们算出了下面的这棵决策树：



假如我们现在发现了一种新物种Python，它是冷血动物，体表带鳞片，并且不是胎生，我们就可以通过这棵决策树来判断它的所属类别。

可以看出，在这个决策过程中，我们一直在对记录的特征进行提问。最初的问题所在的地方叫做**根节点**，在得到结论前的每一个问题都是**中间节点**，而得到的每一个结论（动物的类别）都叫做**叶子节点。**

关键概念：节点
根节点：没有进边，有出边。包含最初的，针对特征的提问。 中间节点：既有进边也有出边，进边只有一条，出边可以有很多条。都是针对特征的提问。 叶子节点：有进边，没有出边，**每个叶子节点都是一个类别标签。** *子节点和父节点：在两个相连的节点中，更接近根节点的是父节点，另一个是子节点。

决策树算法的核心是要解决两个问题：

1）如何从数据表中找出最佳节点和最佳分枝？

2）如何让决策树停止生长，防止过拟合？

几乎所有决策树有关的模型调整方法，都围绕这两个问题展开。这两个问题背后的原理十分复杂，我们会在讲解模型参数和属性的时候为大家简单解释涉及到的部分。在这门课中，我会尽量避免让大家太过深入到决策树复杂的原理和数学公式中（尽管决策树的原理相比其他高级的算法来说是非常简单了），这门课会专注于实践和应用。如果大家希望理解更深入的细节，建议大家在听这门课之前还是先去阅读和学习一下决策树的原理。

1.2 sklearn中的决策树

- 模块sklearn.tree

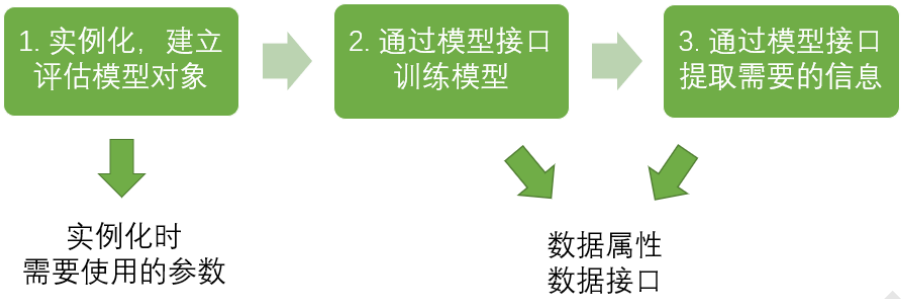
sklearn中决策树的类都在”tree“这个模块之下。这个模块总共包含五个类：

tree.DecisionTreeClassifier	**分类树**
tree.DecisionTreeRegressor	**回归树**
tree.export_graphviz	**将生成的决策树导出为DOT格式，画图专用**
tree.ExtraTreeClassifier	**高随机版本的分类树**
tree.ExtraTreeRegressor	**高随机版本的回归树**

我们会主要讲解分类树和回归树，并用图像呈现给大家。

- sklearn的基本建模流程

在那之前，我们先来了解一下sklearn建模的基本流程。



在这个流程下，分类树对应的代码是：

```
from sklearn import tree                                    #导入需要的模块

clf = tree.DecisionTreeClassifier()                         #实例化
clf = clf.fit(X_train,y_train)                              #用训练集数据训练模型
result = clf.score(X_test,y_test)                           #导入测试集，从接口中调用需要的信息
```

2 DecisionTreeClassifier

class `sklearn.tree.DecisionTreeClassifier` *(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)*

2.1 重要参数

2.1.1 criterion

为了要将表格转化为一棵树，决策树需要找出最佳节点和最佳的分枝方法，对分类树来说，衡量这个“最佳”的指标叫做“不纯度”。通常来说，不纯度越低，决策树对训练集的拟合越好。现在使用的决策树算法在分枝方法上的核心大多是围绕在对某个不纯度相关指标的最优化上。

不纯度基于节点来计算，树中的每个节点都会有一个不纯度，并且子节点的不纯度一定是低于父节点的，也就是说，在同一棵决策树上，叶子节点的不纯度一定是最低的。

Criterion这个参数正是用来决定不纯度的计算方法的。sklearn提供了两种选择：

1）输入”entropy“，使用**信息熵**（Entropy）

2）输入”gini“，使用**基尼系数**（Gini Impurity）

$$Entropy(t) = -\sum_{i=0}^{c-1} p(i|t) log_2 p(i|t)$$

$$Gini(t) = 1 - \sum_{i=0}^{c-1} p(i|t)^2$$

其中t代表给定的节点，i代表标签的任意分类，$p(i|t)$代表标签分类i在节点t上所占的比例。注意，当使用信息熵时，sklearn实际计算的是基于信息熵的信息增益(Information Gain)，即父节点的信息熵和子节点的信息熵之差。

比起基尼系数，信息熵对不纯度更加敏感，对不纯度的惩罚最强。但是**在实际使用中，信息熵和基尼系数的效果基本相同。**信息熵的计算比基尼系数缓慢一些，因为基尼系数的计算不涉及对数。另外，因为信息熵对不纯度更加敏感，所以信息熵作为指标时，决策树的生长会更加"精细"，因此对于高维数据或者噪音很多的数据，信息熵很容易过拟合，基尼系数在这种情况下效果往往比较好。当然，这不是绝对的。

参数	**criterion**
如何影响模型?	确定不纯度的计算方法，帮忙找出最佳节点和最佳分枝，不纯度越低，决策树对训练集的拟合越好
可能的输入有哪些?	不填默认基尼系数，填写gini使用基尼系数，填写entropy使用信息增益
怎样选取参数?	通常就使用基尼系数 数据维度很大，噪音很大时使用基尼系数 维度低，数据比较清晰的时候，信息熵和基尼系数没区别 当决策树的拟合程度不够的时候，使用信息熵 两个都试试，不好就换另外一个

到这里，决策树的基本流程其实可以简单概括如下：

直到没有更多的特征可用，或整体的不纯度指标已经最优，决策树就会停止生长。

- **建立一棵树**

1. **导入需要的算法库和模块**

```
from sklearn import tree
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
```

2. **探索数据**

```
wine = load_wine()

wine.data.shape

wine.target

#如果wine是一张表，应该长这样：
import pandas as pd
pd.concat([pd.DataFrame(wine.data),pd.DataFrame(wine.target)],axis=1)

wine.feature_names
wine.target_names
```

3. **分训练集和测试集**

```
Xtrain, Xtest, Ytrain, Ytest = train_test_split(wine.data,wine.target,test_size=0.3)

Xtrain.shape
Xtest.shape
```

4. **建立模型**

```
clf = tree.DecisionTreeClassifier(criterion="entropy")
clf = clf.fit(Xtrain, Ytrain)
score = clf.score(Xtest, Ytest) #返回预测的准确度

score
```

5. **画出一棵树吧**

```
feature_name = ['酒精','苹果酸','灰','灰的碱性','镁','总酚','类黄酮','非黄烷类酚类','花青素','颜色强度','色调','od280/od315稀释葡萄酒','脯氨酸']

import graphviz
dot_data = tree.export_graphviz(clf
                                ,feature_names= feature_name
                                ,class_names=["琴酒","雪莉","贝尔摩德"]
                                ,filled=True
                                ,rounded=True
                               )
graph = graphviz.Source(dot_data)
graph
```

6. **探索决策树**

```
#特征重要性
clf.feature_importances_

[*zip(feature_name,clf.feature_importances_)]
```

我们已经在只了解一个参数的情况下，建立了一棵完整的决策树。但是回到步骤4建立模型，score会在某个值附近波动，引起步骤5中画出来的每一棵树都不一样。它为什么会不稳定呢？如果使用其他数据集，它还会不稳定吗？

我们之前提到过，无论决策树模型如何进化，在分枝上的本质都还是追求某个不纯度相关的指标的优化，而正如我们提到的，不纯度是基于节点来计算的，也就是说，决策树在建树时，是靠优化节点来追求一棵优化的树，但最优的节点能够保证最优的树吗？集成算法被用来解决这个问题：sklearn表示，既然一棵树不能保证最优，那就建更多的不同的树，然后从中取最好的。怎样从一组数据集中建不同的树？在每次分枝时，不从使用全部特征，而是随机选取一部分特征，从中选取不纯度相关指标最优的作为分枝用的节点。这样，每次生成的树也就不同了。

```
clf = tree.DecisionTreeClassifier(criterion="entropy",random_state=30)
clf = clf.fit(Xtrain, Ytrain)
score = clf.score(Xtest, Ytest) #返回预测的准确度

score
```

2.1.2 random_state & splitter

random_state用来设置分枝中的随机模式的参数，默认None，在高维度时随机性会表现更明显，低维度的数据（比如鸢尾花数据集），随机性几乎不会显现。输入任意整数，会一直长出同一棵树，让模型稳定下来。

splitter也是用来控制决策树中的随机选项的，有两种输入值，输入”best"，决策树在分枝时虽然随机，但是还是会优先选择更重要的特征进行分枝（重要性可以通过属性feature_importances_查看），输入“random"，决策树在分枝时会更加随机，树会因为含有更多的不必要信息而更深更大，并因这些不必要信息而降低对训练集的拟合。这也是防止过拟合的一种方式。当你预测到你的模型会过拟合，用这两个参数来帮助你降低树建成之后过拟合的可能性。当然，树一旦建成，我们依然是使用剪枝参数来防止过拟合。

```
clf = tree.DecisionTreeClassifier(criterion="entropy"
                                  ,random_state=30
                                  ,splitter="random"
                                 )
clf = clf.fit(Xtrain, Ytrain)
score = clf.score(Xtest, Ytest)

score

import graphviz
dot_data = tree.export_graphviz(clf
                                ,feature_names= feature_name
                                ,class_names=["琴酒","雪莉","贝尔摩德"]
                                ,filled=True
                                ,rounded=True
                               )
graph = graphviz.Source(dot_data)
graph
```

2.1.3 剪枝参数

在不加限制的情况下，一棵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，这就是说，**它会在训练集上表现很好，在测试集上却表现糟糕。**我们收集的样本数据不可能和整体的状况完全一致，因此当一棵决策树对训练数据有了过于优秀的解释性，它找出的规则必然包含了训练样本中的噪声，并使它对未知数据的拟合程度不足。

```
#我们的树对训练集的拟合程度如何?
score_train = clf.score(Xtrain, Ytrain)
score_train
```

为了让决策树有更好的泛化性，我们要对决策树进行剪枝。**剪枝策略对决策树的影响巨大，正确的剪枝策略是优化决策树算法的核心。**sklearn为我们提供了不同的剪枝策略：

- **max_depth**

限制树的最大深度，超过设定深度的树枝全部剪掉

这是用得最广泛的剪枝参数，在高维度低样本量时非常有效。决策树多生长一层，对样本量的需求会增加一倍，所以限制树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=3开始尝试，看看拟合的效果再决定是否增加设定深度。

- **min_samples_leaf & min_samples_split**

min_samples_leaf限定，一个节点在分枝后的每个子节点都必须包含至少min_samples_leaf个训练样本，否则分枝就不会发生，或者，分枝会朝着满足每个子节点都包含min_samples_leaf个样本的方向去发生

一般搭配max_depth使用，在回归树中有神奇的效果，可以让模型变得更加平滑。这个参数的数量设置得太小会引起过拟合，设置得太大就会阻止模型学习数据。一般来说，建议从=5开始使用。如果叶节点中含有的样本量变化很大，建议输入浮点数作为样本量的百分比来使用。同时，这个参数可以保证每个叶子的最小尺寸，可以在回归问题中避免低方差，过拟合的叶子节点出现。对于类别不多的分类问题，=1通常就是最佳选择。

min_samples_split限定，一个节点必须要包含至少min_samples_split个训练样本，这个节点才允许被分枝，否则分枝就不会发生。

```
clf = tree.DecisionTreeClassifier(criterion="entropy"
                                  ,random_state=30
                                  ,splitter="random"
                                  ,max_depth=3
                                  ,min_samples_leaf=10
                                  ,min_samples_split=10
                                 )

clf = clf.fit(Xtrain, Ytrain)

dot_data = tree.export_graphviz(clf
                                ,feature_names= feature_name
                                ,class_names=["琴酒","雪莉","贝尔摩德"]
                                ,filled=True
                                ,rounded=True
                               )
graph = graphviz.Source(dot_data)
```

```
graph

clf.score(Xtrain,Ytrain)
clf.score(Xtest,Ytest)
```

- **max_features & min_impurity_decrease**

一般max_depth使用，用作树的"精修"

max_features限制分枝时考虑的特征个数，超过限制个数的特征都会被舍弃。和max_depth异曲同工，max_features是用来限制高维度数据的过拟合的剪枝参数，但其方法比较暴力，是直接限制可以使用的特征数量而强行使决策树停下的参数，在不知道决策树中的各个特征的重要性的情况下，强行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。

min_impurity_decrease限制信息增益的大小，信息增益小于设定数值的分枝不会发生。这是在0.19版本种更新的功能，在0.19版本之前时使用min_impurity_split。

- 确认最优的剪枝参数

那具体怎么来确定每个参数填写什么值呢？这时候，我们就要使用确定超参数的曲线来进行判断了，继续使用我们已经训练好的决策树模型clf。超参数的学习曲线，是一条以超参数的取值为横坐标，模型的度量指标为纵坐标的曲线，它是用来衡量不同超参数取值下模型的表现的线。在我们建好的决策树里，我们的模型度量指标就是score。

```
import matplotlib.pyplot as plt

test = []
for i in range(10):
    clf = tree.DecisionTreeClassifier(max_depth=i+1
                                      ,criterion="entropy"
                                      ,random_state=30
                                      ,splitter="random"
                                     )
    clf = clf.fit(Xtrain, Ytrain)
    score = clf.score(Xtest, Ytest)
    test.append(score)
plt.plot(range(1,11),test,color="red",label="max_depth")
plt.legend()
plt.show()
```

思考：

1. 剪枝参数一定能够提升模型在测试集上的表现吗？ - 调参没有绝对的答案，一切都是看数据本身。
2. 这么多参数，一个个画学习曲线？

无论如何，剪枝参数的默认值会让树无尽地生长，这些树在某些数据集上可能非常巨大，对内存的消耗。所以如果你手中的数据集非常大，你已经预测到无论如何你都是要剪枝的，那提前设定这些参数来控制树的复杂性和大小会比较好。

2.1.4 目标权重参数

- **class_weight & min_weight_fraction_leaf**

完成样本标签平衡的参数。样本不平衡是指在一组数据集中，标签的一类天生占有很大的比例。比如说，在银行要判断“一个办了信用卡的人是否会违约”，就是是vs否（1%：99%）的比例。这种分类状况下，即便模型什么也不做，全把结果预测成“否”，正确率也能有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。该参数默认None，此模式表示自动给与数据集中的所有标签相同的权重。

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标准（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来更容易优化树结构，这确保叶节点至少包含样本权重的总和的一小部分。

2.2 重要属性和接口

属性是在模型训练之后，能够调用查看的模型的各种性质。对决策树来说，最重要的是feature_importances_，能够查看各个特征对模型的重要性。

sklearn中许多算法的接口都是相似的，比如说我们之前已经用到的fit和score，几乎对每个算法都可以使用。除了这两个接口之外，决策树最常用的接口还有apply和predict。apply中输入测试集返回每个测试样本所在的叶子节点的索引，predict输入测试集返回每个测试样本的标签。返回的内容一目了然并且非常容易，大家感兴趣可以自己下去试试看。

```
#apply返回每个测试样本所在的叶子节点的索引
clf.apply(Xtest)

#predict返回每个测试样本的分类/回归结果
clf.predict(Xtest)
```

至此，我们已经学完了分类树DecisionTreeClassifier和用决策树绘图（export_graphviz）的所有基础。我们讲解了决策树的基本流程，分类树的七个参数，一个属性，四个接口，以及绘图所用的代码。

七个参数：Criterion，两个随机性相关的参数（random_state，splitter），四个剪枝参数（max_depth,，min_sample_leaf，max_feature，min_impurity_decrease）

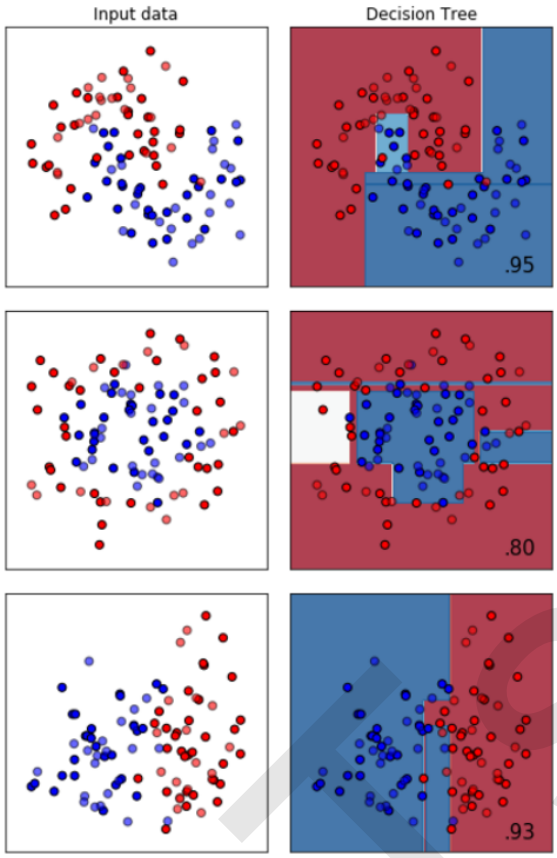
一个属性：feature_importances_

四个接口：fit，score，apply，predict

有了这些知识，基本上分类树的使用大家都能够掌握了，接下来再到实例中去磨练就好。

2.3 实例：分类树在合成数集上的表现

我们在红酒数据集上画出了一棵树，并且展示了多个参数会对树形成这样的影响，接下来，我们将在不同结构的数据集上测试一下决策树的效果，让大家更好地理解决策树。



1. **导入需要的库**

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.tree import DecisionTreeClassifier
```

2. **生成三种数据集**

我们先从sklearn自带的数据库中生成三种类型的数据集：1）月亮型数据，2）环形数据，3）二分型数据

```
#make_classification库生成随机的二分型数据
X, y = make_classification(n_samples=100, #生成100个样本
                           n_features=2,  #包含2个特征，即生成二维数据
                           n_redundant=0, #添加冗余特征0个
                           n_informative=2, #包含信息的特征是2个
                           random_state=1,  #随机模式1
                           n_clusters_per_class=1 #每个簇内包含的标签类别有1个
                          )

#在这里可以查看一下X和y，其中X是100行带有两个2特征的数据，y是二分类标签
#也可以画出散点图来观察一下X中特征的分布
#plt.scatter(X[:,0],X[:,1])
```

```
#从图上可以看出，生成的二分型数据的两个簇离彼此很远，这样不利于我们测试分类器的效果，因此我们使用np生成随机数组，通过让已经生成的二分型数据点加减0~1之间的随机数，使数据分布变得更散更稀疏
#注意，这个过程只能够运行一次，因为多次运行之后X会变得非常稀疏，两个簇的数据会混合在一起，分类器的效应会继续下降
rng = np.random.RandomState(2) #生成一种随机模式
X += 2 * rng.uniform(size=X.shape) #加减0~1之间的随机数
linearly_separable = (X, y) #生成了新的X，依然可以画散点图来观察一下特征的分布
#plt.scatter(X[:,0],X[:,1])

#用make_moons创建月亮型数据，make_circles创建环形数据，并将三组数据打包起来放在列表datasets中
datasets = [make_moons(noise=0.3, random_state=0),
            make_circles(noise=0.2, factor=0.5, random_state=1),
            linearly_separable]
```

3. **画出三种数据集和三棵决策树的分类效应图像**

```
#创建画布，宽高比为6*9
figure = plt.figure(figsize=(6, 9))
#设置用来安排图像显示位置的全局变量i
i = 1

#开始迭代数据，对datasets中的数据进行for循环

for ds_index, ds in enumerate(datasets):

    #对X中的数据进行标准化处理，然后分训练集和测试集
    X, y = ds
    X = StandardScaler().fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4,
random_state=42)

    #找出数据集中两个特征的最大值和最小值，让最大值+0.5，最小值-0.5，创造一个比两个特征的区间本身更大一点的区间
    x1_min, x1_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    x2_min, x2_max = X[:, 1].min() - .5, X[:, 1].max() + .5

    #用特征向量生成网格数据，网格数据，其实就相当于坐标轴上无数个点
    #函数np.arange在给定的两个数之间返回均匀间隔的值，0.2为步长
    #函数meshgrid用以生成网格数据，能够将两个一维数组生成两个二维矩阵。
    #如果第一个数组是narray，维度是n，第二个参数是marray，维度是m。那么生成的第一个二维数组是以narray为行，m行的矩阵，而第二个二维数组是以marray的转置为列，n列的矩阵
    #生成的网格数据，是用来绘制决策边界的，因为绘制决策边界的函数contourf要求输入的两个特征都必须是二维的
    array1,array2 = np.meshgrid(np.arange(x1_min, x1_max, 0.2),
                        np.arange(x2_min, x2_max, 0.2))

    #接下来生成彩色画布
    #用ListedColormap为画布创建颜色，#FF0000正红，#0000FF正蓝
    cm = plt.cm.RdBu
    cm_bright = ListedColormap(['#FF0000', '#0000FF'])

    #在画布上加上一个子图，数据为len(datasets)行，2列，放在位置i上
    ax = plt.subplot(len(datasets), 2, i)
```

```
    #到这里为止，已经生成了0~1之间的坐标系3个了，接下来为我们的坐标系放上标题
    #我们有三个坐标系，但我们只需要在第一个坐标系上有标题，因此设定if ds_index==0这个条件
    if ds_index == 0:
        ax.set_title("Input data")

    #将数据集的分布放到我们的坐标系上
    #先放训练集
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train,
               cmap=cm_bright,edgecolors='k')
    #放测试集
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test,
               cmap=cm_bright, alpha=0.6,edgecolors='k')

     #为图设置坐标轴的最大值和最小值，并设定没有坐标轴
    ax.set_xlim(array1.min(), array1.max())
    ax.set_ylim(array2.min(), array2.max())
    ax.set_xticks(())
    ax.set_yticks(())

    #每次循环之后，改变i的取值让图每次位列不同的位置
    i += 1

    #至此为止，数据集本身的图像已经布置完毕，运行以上的代码，可以看见三个已经处理好的数据集

    ##############################从这里开始是决策树模型##########################

    #迭代决策树，首先用subplot增加子图，subplot(行，列，索引)这样的结构，并使用索引i定义图的位置
    #在这里，len(datasets)其实就是3，2是两列
    #在函数最开始，我们定义了i=1，并且在上边建立数据集的图像的时候，已经让i+1,所以i在每次循环中的取值是2，4，6
    ax = plt.subplot(len(datasets),2,i)

    #决策树的建模过程：实例化 → fit训练 → score接口得到预测的准确率
    clf = DecisionTreeClassifier(max_depth=5)
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)

    #绘制决策边界，为此，我们将为网格中的每个点指定一种颜色[x1_min，x1_max] x [x2_min，x2_max]
    #分类树的接口，predict_proba，返回每一个输入的数据点所对应的标签类概率
    #类概率是数据点所在的叶节点中相同类的样本数量/叶节点中的样本总数量
    #由于决策树在训练的时候导入的训练集X_train里面包含两个特征，所以我们在计算类概率的时候，也必须导入结构相同的数组，即是说，必须有两个特征
    #ravel()能够将一个多维数组转换成一维数组
    #np.c_是能够将两个数组组合起来的函数
    #在这里，我们先将两个网格数据降维降维成一维数组，再将两个数组链接变成含有两个特征的数据，再带入决策树模型，生成的Z包含数据的索引和每个样本点对应的类概率，再切片，且出类概率
    Z = clf.predict_proba(np.c_[array1.ravel(),array2.ravel()])[:, 1]

    #np.c_[np.array([1,2,3]), np.array([4,5,6])]

    #将返回的类概率作为数据，放到contourf里面绘制去绘制轮廓
    Z = Z.reshape(array1.shape)
```

```
        ax.contourf(array1, array2, Z, cmap=cm, alpha=.8)

        #将数据集的分布放到我们的坐标系上
        # 将训练集放到图中去
        ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright,
                   edgecolors='k')
        # 将测试集放到图中去
        ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright,
                   edgecolors='k', alpha=0.6)

        #为图设置坐标轴的最大值和最小值
        ax.set_xlim(array1.min(), array1.max())
        ax.set_ylim(array2.min(), array2.max())
        #设定坐标轴不显示标尺也不显示数字
        ax.set_xticks(())
        ax.set_yticks(())

        #我们有三个坐标系，但我们只需要在第一个坐标系上有标题，因此设定if ds_index==0这个条件
        if ds_index == 0:
            ax.set_title("Decision Tree")

        #写在右下角的数字
        ax.text(array1.max() - .3, array2.min() + .3, ('{:.1f}%'.format(score*100)),
                size=15, horizontalalignment='right')

        #让i继续加一
        i += 1

    plt.tight_layout()
    plt.show()
```

从图上来看，每一条线都是决策树在二维平面上画出的一条决策边界，每当决策树分枝一次，就有一条线出现。当数据的维度更高的时候，这条决策边界就会由线变成面，甚至变成我们想象不出的多维图形。

同时，很容易看得出，分类树天生不擅长环形数据。每个模型都有自己的决策上限，所以一个怎样调整都无法提升表现的可能性也是有的。当一个模型怎么调整都不行的时候，我们可以选择换其他的模型使用，不要在一棵树上吊死。顺便一说，最擅长月亮型数据的是最近邻算法，RBF支持向量机和高斯过程；最擅长环形数据的是最近邻算法和高斯过程；最擅长对半分的数据的是朴素贝叶斯，神经网络和随机森林。