sklearn

why do we learn sklearn?

在 ML 中，sklearn 是非常重要的，可以调用各种各样的算法，一些算法我们是写不出来的，那么我们就可以用 sklearn 来调用那些算法，从而进行数据分析。当然，我们也要明白一些参数的作用，了解参数的作用最好是去了解算法的推到过程。

Decision tree

概述

决策树是如何工作的?

- 决策树是一种非参数的有监督学习方法。什么是非参数？就是说什么样的数据类型都可以，比如说 boolean, int, char,string,都可以。supervise 就是要 label。决策树能够从一系列有特征和标签的数据中总结出决策规则，并用树状的结构来呈现这些规则，以解决分类和回归的问题。

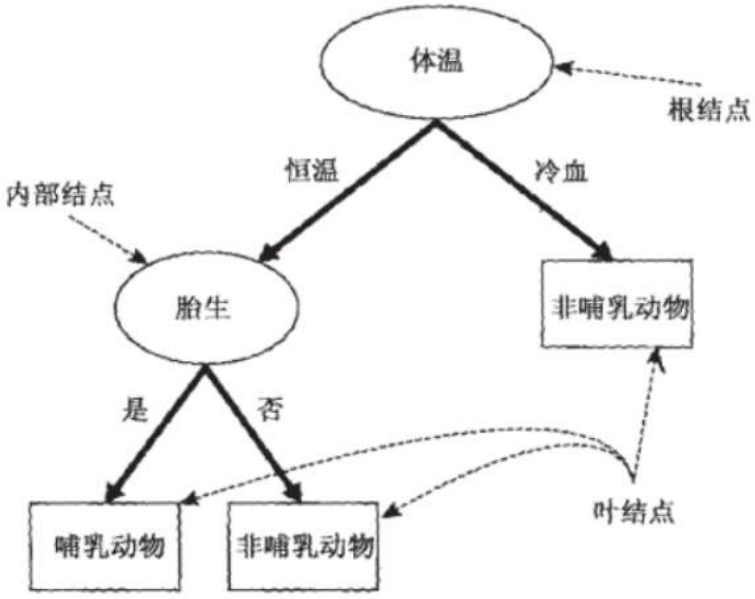
- 决策树算法很容易理解，适用各种数据，在解决各种问题时都有良好的表现，尤其是以树模型为核心的各种集成算法，在各个行业和领域都有广泛的应用

- 我们来了解一些决策树是如何工作的。决策树算法的本质就是一种图结构，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，我们看看这个数据集，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
巨蟒	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

- 我们现在的目标是讲动物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这棵决策树



- 假设我们现在有一个新物种，我们就可以通过这些判断，决策树判断，来知晓这个物种是属于哺乳还是非哺乳

- 在决策的过程中，我们一直在对记录的特征进行提问。最初问题所在的地方叫做根节点(root),在得到结论前的没一个问题都是中间节点(decision node)，而得到的每一个结论都叫做叶子节点（leaf)

- 一些很简单的概念

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

  - 如何从数据表中找出最佳节点和最佳分支（branch）？也就是说根节点应该选哪个啊，选完之后下一个的 decision node 应该是什么？

  - 如何让决策树停止生长，防止过拟合？就是说一定要用完所有的 variables 吗？使用部分行不行？

决策树算法推导

- 我们在数据中会看到有非常多的属性，但决策树要求我们不断地去选节点，怎么选呢？用什么来衡量呢？熵和基尼是一样的，公式不一样，但是总体是一样的
  - Entropy （熵）
    - 衡量一个节点的不确定性
    - 公式，Pk 的意思是某一个划分的概率

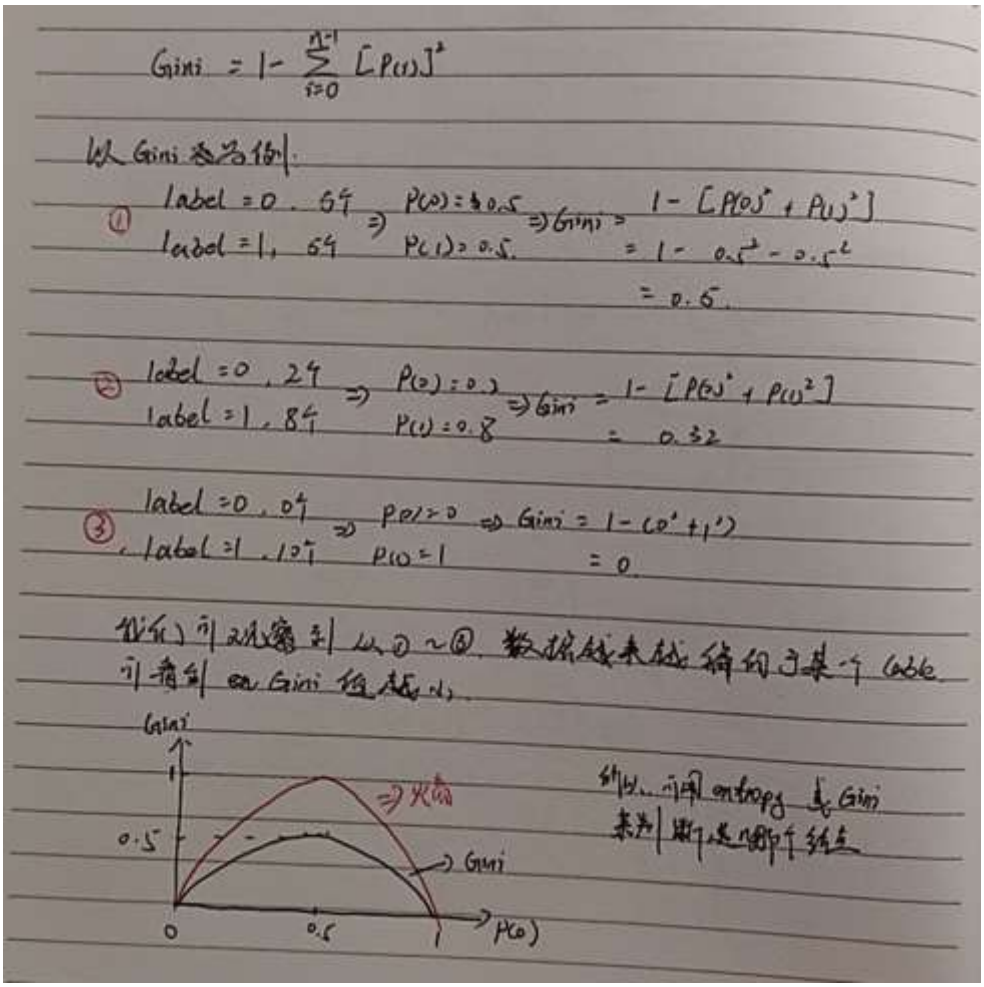
$$\mathbf{H(X)} = -\sum_{k=1}^{N} p_k \log_2(p_k)$$

    - 当熵值低的时候，物体非常有秩序，也可以说当前节点内，可以把 label 划分好

STRONG CONSISTENCY IN THE NODE
LOW ENTROPY
高秩序度
低熵

LOW CONSISTENCY IN THE NODE
HIGH ENTROPY
低秩序度
高熵

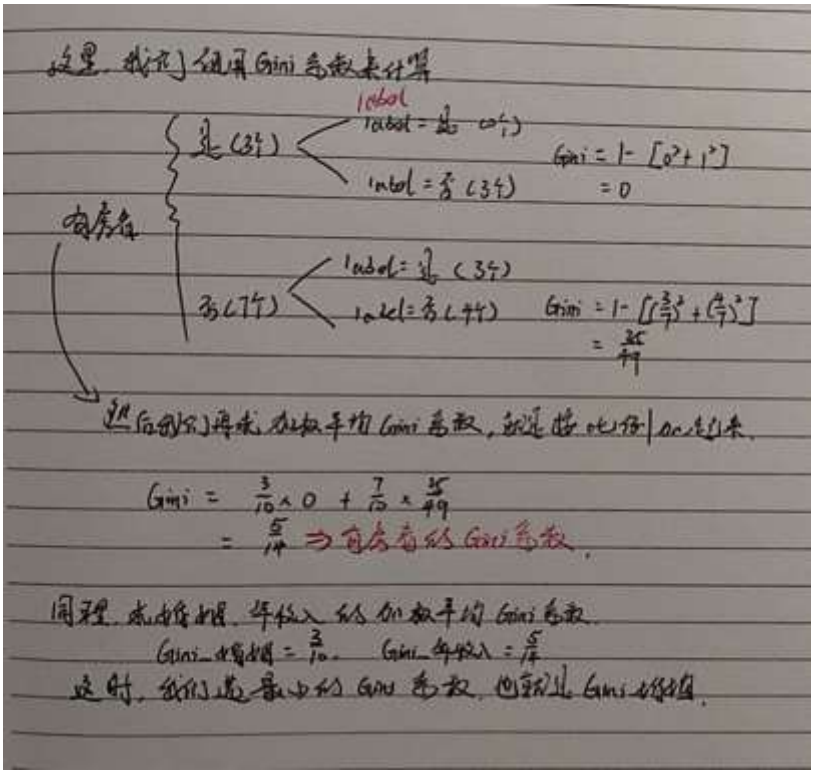
  - Gini（基尼）
    - 公式
    - 这个 pi 依旧是概率的意思
- 可以去了解 ID3 算法，C4.5 算法，CART 算法
- example
  - 例子 1



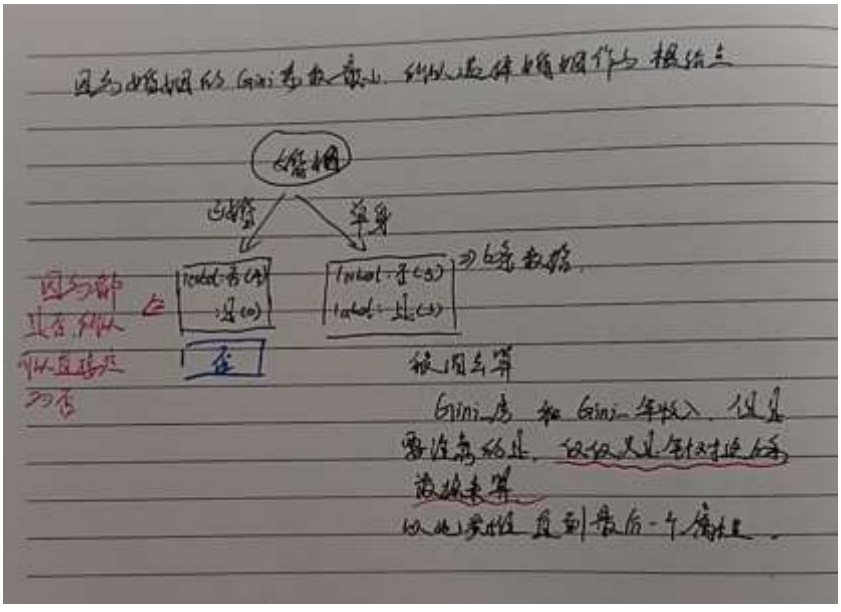
• 例子 2，假设我们有这样的数据，我们要去计算哪个属性可以作为根节点，哪个属性可以作为下一个节点

				label
Id	有房者	婚姻	年收入	拖欠贷款?
1	是	单身	125k	否
2	否	已婚	100k	否
3	否	单身	70k	否
4	是	已婚	120k	否
5	否	已婚	95k	是
6	否	已婚	60k	否
7	是	已婚	220k	否
8	否	单身	85k	是
9	否	已婚	75k	否
10	否	单身	90k	是

• 我们要先选根节点，我们就计算哪个 Gini 系数最小，选最小的。关于年收入，要自己去划分，这里的划分点是<80k 或者 >=80k



- 这个过程呢，或减少一些数据，这个是计算过程应该注意的



sklearn 中的决策树

- 模块 sklearn.tree
  - sklearn 中决策树的类都在 tree 这个模块下，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

    - 在这里，我们会学习分类树和回归树

- sklearn 的基本建模流程

    - 就是三步走

    - 对应代码

```
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
```

- 当我们了解 entropy 和 gini 的时候，我们就可以简单地概括决策树的流程，

  每次计算，都会得出 entropy 或者 gini 的值，选最小的。知道没有更多的特征可以用，或者整体不纯度指标达到最优，决策树停止生长

DecisionTreeClassifier 与红酒数据集

重要参数,我们可以看到，这里面有非常多的参数，我们要去理解这些都是什么意思

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*)

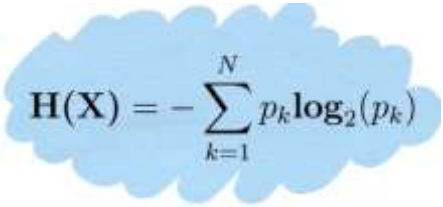
- criterion

  - 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上的核心大多是围绕在对某个不纯度的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，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 提供了两个选择

    - entropy



$$\mathbf{H(X)} = -\sum_{k=1}^{N} p_k \log_2(p_k)$$

    - gini

  - 比起基尼系数，熵对不纯度更加敏感，对不纯度的惩罚最强，因为有个 log。但是在实际使用中，entropy 和 gini 的效果基本上差不多。熵的计算要慢一点，因为有 log。另外因为熵对不纯度更加敏感，所以熵作为指标时，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，熵很容易过拟合。而 Gini 系数在这种情况下会好一点。所以，当模型拟合程度不足的时候，就是在训练集和测试集表现一般般的时候，可以使用熵，但也不是绝对。

  - summary

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

- random_state & splitter

- random_state

  - 当我们多次运行的时候，我们会发现 accuracy 可能每次都不太一样，上下波动，这样就会导致每次决策树画出来都是不一样的。我们决策树的思想的本质就是追求某个不纯度相关的指标的优化，而不纯度是基于节点来计算的。但是最优的节点能保证最优的树吗？因为每次都会变，每次的树都不一样。

```
#use entropy as criterion
clf = tree.DecisionTreeClassifier(criterion='entropy' random_state=5)
#fit the data
clf.fit(Xtrain, ytrain)
#check the accuracy
score = clf.score(Xtest, ytest)
print("The accuracy is: {0:2.5f}".format(score))
```

```
The accuracy is: 0.94444
```

  - 在高纬度时，这个随机性会更加明显

  - 于是，我们采用 random_state 来解决这个问题。既然一棵树不能保证最优，那就多建几棵不同的树，然后从中选取最好的。那怎么样从一组数据中建立不同的树呢？在每次分枝的时候，不选用全部特征，随机选取一部分特征，进行决策树的过程，这样，每次生成的树也就不一样啦。

  - random_state = 任意整数。这就是控制随机的参数，默认是 None。比如说，用了这个参数和没用这个参数的区别。

    - 没用这个参数

      - 每次生成的结果都不一样，这次关了，下次再重新运行，结果又不一样

    - 用了这个参数

      - 结果一定是一致的，不会再变。用了这个参数会让模型稳定下来

- splitter

  - splitter 也是用来控制决策树中的随机选项的，有两个输入值，‘best’， ‘random’

    - 当 splitter = ’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

- 当 splitter = ‘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的信息而更深更大，并因这些不必要的信息而降低训练集的拟合。这也是防止过拟合的一种方式

- 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，或者到没有更多的特征可用为止。这样的决策树往往会过拟合。就是说，在训练集上表现很好，但是在测试集上却表现很差。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

  - 为了让决策树有更好的泛化性，我们需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

    - max_depth

      - 限制树的最大深度，超过设定深度的树枝全部剪掉。

      - 这个参数是使用最广泛的剪枝参数，特别是在高纬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，呈指数级增长。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 3 开始，看看拟合效果再决定是否增加深度

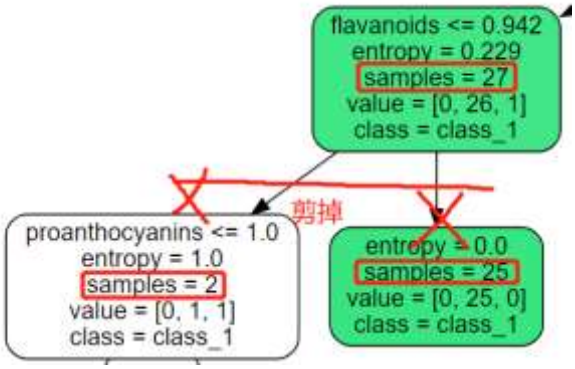
    - min_samples_leaf & min_samples_split

      - 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。如果叶子节点中含有样本数量变化很大，建议输入浮点数作为样本量的百分比来使用。比如说，输入 0.05，那么每次就按分枝前 sample 的 5%这个比例走。

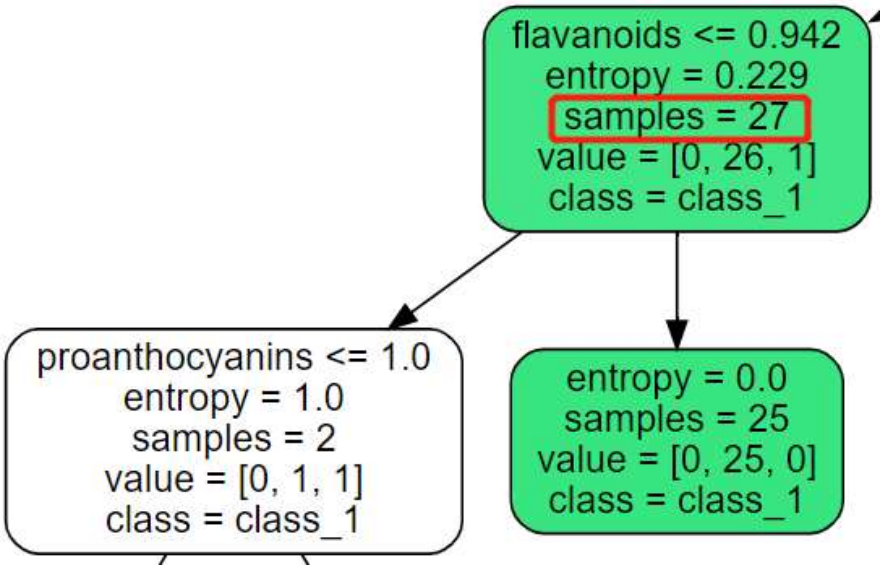
- 同时，这个参数可以保证每个叶子的最小尺寸，可以在回归问题上避免低方差，过拟合的叶子节点出现。对于类别不多的分类问题，=1 通常就是最佳选择

- 例子，现在有这么一种情况，如图所示，然后我们设置 min_samples_leaf=5,但是我们看到白色框框那个的 samples=2，明显 < 5，我们要把这一层给剪掉，尽管它旁边的绿色框有 samples=25。



- min_sample_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

- 例子，假设我设置 min_samples_split = 30,当它遇到 samples<30 时，就会停止 split。比如说这个 27，就停在 27 那一层，不会再继续 split 了



- max_features & min_impurity_decrease

- max_features

- 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

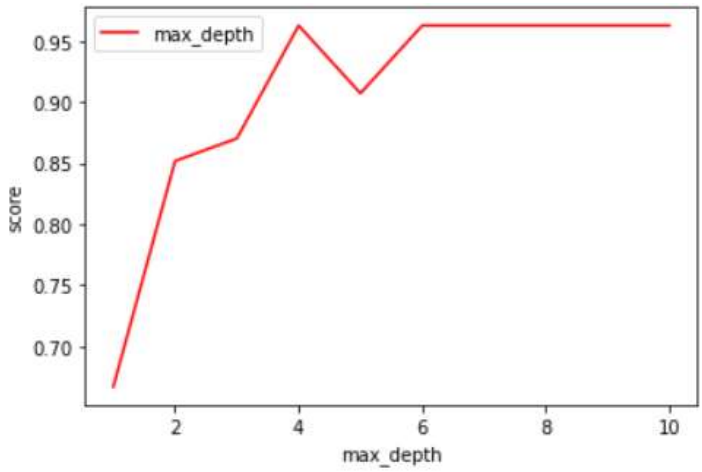
- max_features 是用来限制高维数据的过拟合的剪枝参数，但其方法比较暴力，是直接限制可以使用的特征数量而强行使决策树停下的参数，在不知道决策树中的各个特征的重要性的情况下，强行设定这个参数可能会导致模型学习不足。如果希望通过降维的方式防止过拟合，建议使用 PCA，ICA 或者特征选择模块中的降维算法

  - min_impurity_decrease

    - 限制熵增益（entropy gain）的大小。信息增益小于设定数值的分枝就不会发生。父节点的 entropy 一定大于子节点的 entropy。差值越大，说明对决策树的贡献越大

- 确认最优剪枝参数

  - 那具体是怎么来确定每个参数的值呢？这时候，我们就要用确定超参数的曲线来进行判断了。就是我们给定一定范围的值，或者给定一些值，在 for 循环里面不断运行，我们找出表现最后的点即可。模型度量指标就是准确率



  - 思考？

    - 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

    - 这么多参数，一个个地画曲线吗？

      - 下面在泰坦尼克号案例中会解答

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

- class_weight & min_weight_fraction_leaf

  - 这是目标权重参数。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%）。假设是这个比例。这种分类情况下，即便模型什么也不做，全把结果预测成了“否”，正确率也能有 99%。但是对银行来说，更偏向于那 1%的结果。因此我们要使用 class_weight 参数来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。该参数默认是 None，此模式表示自动给与数据集中所有标签相同的权重

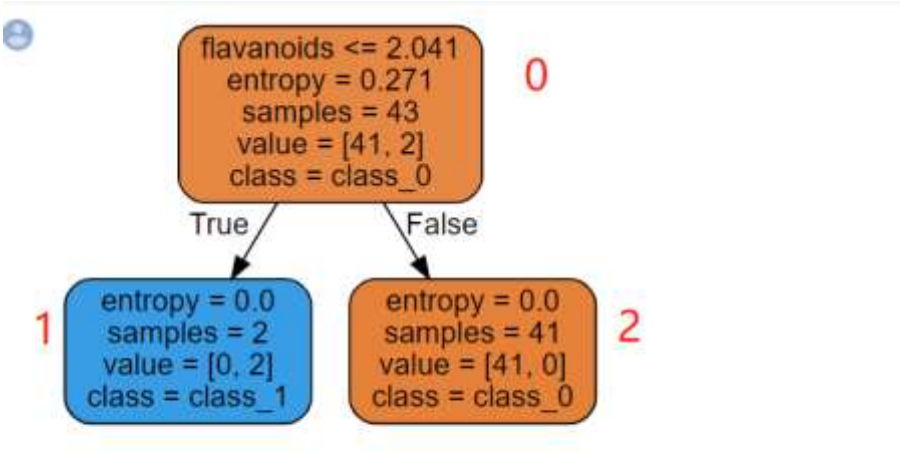
  - 有了权重之后，样本量就不再是单纯地记录数目了，而是受输入的权重影响了，因此，这时候剪枝，就需要搭配 min_weight_fraction_leaf 这个基于权重剪枝的参数来使用。另外注意，基于权重的剪枝参数（例如：min_weight_fraction_leaf）将比不知道样本权重的标准（比如 min_samples_leaf）更少偏向主导类。如果样本是加权的，则使用基于权重的预修剪标准来更容易优化树结构，这确保叶节点至少包含样本权重的总和的一小部分。

重要属性和接口

- 属性是在模型训练之后，能够调用查看模型的各种性质。对决策树来说，最重要的就是 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。除了这个两个接口以外，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




```
61] clf.apply(Xitest)

    array([2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

- predict 输入测试集返回每个测试样本的标签

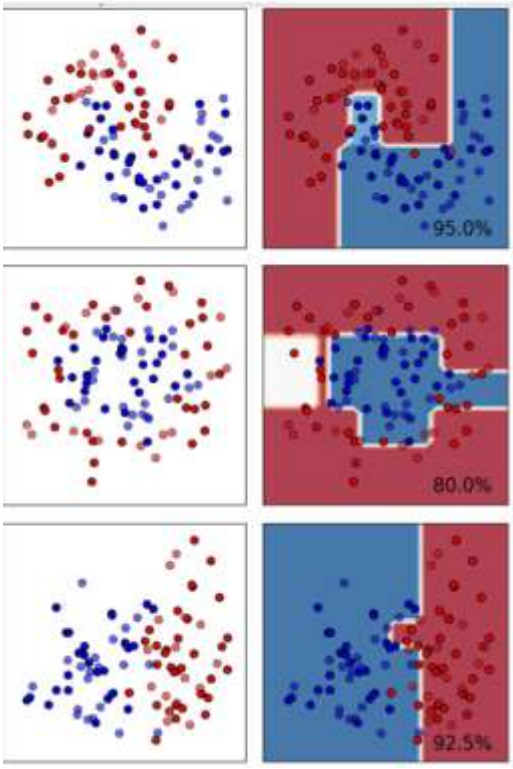
```
clf.predict(Xitest)

array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

- 在这里得说一下，所有接口中要求输入 X_train 和 X_test 的部分，输入的特征矩阵必须至少是一个二维矩阵。sklearn 不接受任何以为矩阵作为特征矩阵被输入。如果你的数据的确只有一特征，那必须使用 reshape（-1,1）来给矩阵增维。如果你的数据只有一个特征和一个样本，使用 reshape（1.-1）来给数据增维

其他形状的数据集

- 我们看到决策树不擅长做环形数据，也就是第二行。所以，当数据是环形的时候，需要考虑其他的。



- 最擅长月亮型数据（第一行）的是 KNN，RBF SVM，和高斯过程

- 最擅长环形数据的是 KNN 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

DecisionTreeRegressor

重要参数，属性以及接口

class `sklearn.tree.DecisionTreeRegressor` *(criterion='mse', 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, presort=False)*

- 我们可以看到，参数几乎跟分类树差不多。需要注意的是，在回归树中，没有标签分布是否均衡的问题，因为是最后预测的不是标签，而是具体数字，因此没有 class_weight 这样的参数

- criterion

  - 输入‘mse’，mean squared error(MSE)

- 父节点和叶子节点之间的均方误差的差额将被用来作为特征选择的标准，这种方法通过使用叶子节点的均值来最小化 L2 损失

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

- 其中 N 是样本数量，i 是每一个数据样本，fi 是模型回归出的数值，yi 是样本点 i 实际的数值标签。所以 MSE 的本质，其实是样本真实数据与回归结果的差异。在回归树中，MSE 不只是我们的分枝质量衡量指标，也是我们最常用的衡量回归树回归质量的指标，当我们在使用交叉验证，或者其他方式获取回归树的结果时，我们往往选择均方误差作为我们的评估（在分类树中这个指标是 score 代表的预测准确率）。在回归中，我们追求的是，MSE 越小越好。

- 然而，回归树的接口 score 返回的是 R 平方，并不是 MSE。R 平方被定义如下：

$$R^2 = 1 - \frac{u}{v}$$

$$u = \sum_{i=1}^{N}(f_i - y_i)^2 \qquad v = \sum_{i=1}^{N}(y_i - \hat{y})^2$$

  - 其中 u 是残差平方和（MSE * N），v 是总平和，N 是样本数量，i 是每一个数据样本，fi 是模型回归出的数值，yi 是样本点 i 实际的数值标签。y 帽是真实数值标签的平均数。R 平方可以为正为负（如果模型的残差平方和远远大于模型的总平方和，模型非常糟糕，R 平方就会为负），而均方误差永远为正。

- 值得一提的是，虽然均方误差永远为正，但是 sklearn 当中使用均方误差作为评判标准时，却是计算”负均方误差“（neg_mean_squared_error）.这是因为 sklearn 在计算模型评估指标的时候，会考虑指标本身的性质，均方误差本身是一种误差，所以被 sklearn 划分为模型的一种损失(loss)，因此在 sklearn 当中，都以负数表示。真正的均方误差 MSE 的数值，其实就是 neg_mean_squared_error 去掉负号的数字。

- 输入‘friedman_mse’

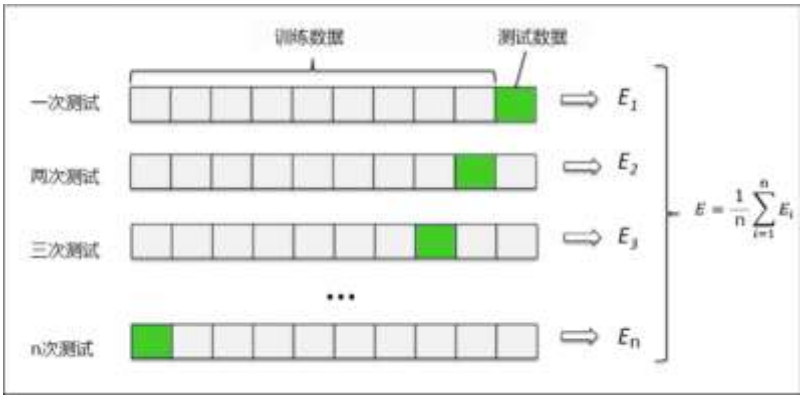
  - 使用费尔德曼均方误差，这种指标使用弗里德曼针对潜在分枝中的问题改进后的均方误差

- 输入‘mae’，MAE（mean absolute error）

  - 这种指标使用叶节点的中值来最小化 L1 损失

- 交叉验证，cross validation

  • 这是用来观察模型的稳定性的一种方法，我们将数据划分为 n 份，依次使用其中一份作为测试集，其他 n-1 份作为训练集，多次计算模型的精确性来评估模型的平均准确程度。训练集和测试集的划分会干扰模型的结果，因此用交叉验证 n 次的结果求出的平均值，是对模型效果的一个更好的度量。



决策树优缺点

优点

- 易于理解和解释，因为树木可以画出来被看见

- 需要很少的数据准备。其他很多算法通常都需要数据规范化，需要创建虚拟变量并删除空值等。但请注意，sklearn 中的决策树模块不支持对缺失值的处理。

- 使用树的成本（比如说，在预测数据的时候）是用于训练树的数据点的数量的对数，相比于其他算法，这是一个很低的成本。

- 能够同时处理数字和分类数据，既可以做回归又可以做分类。其他技术通常专门用于分析仅具有一种变量类型的数据集。

- 能够处理多输出问题，即含有多个标签的问题，注意与一个标签中含有多种标签分类的问题区别开

- 是一个白盒模型，结果很容易能够被解释。如果在模型中可以观察到给定的情况，则可以通过布尔逻辑轻松解释条件。相反，在黑盒模型中（例如，在

人工神经网络中），结果可能更难以解释。

- 可以使用统计测试验证模型，这让我们可以考虑模型的可靠性。

- 即使其假设在某种程度上违反了生成数据的真实模型，也能够表现良好。

缺点

- 决策树学习者可能创建过于复杂的树，这些树不能很好地推广数据。这称为过度拟合。修剪，设置叶节点所需的最小样本数或设置树的最大深度等机制是避免此问题所必需的，而这些参数的整合和调整对初学者来说会比较晦涩

- 决策树可能不稳定，数据中微小的变化可能导致生成完全不同的树，这个问题需要通过集成算法来解决

- 决策树的学习是基于贪婪算法，它靠优化局部最优（每个节点的最优）来试图达到整体的最优，但这种做法不能保证返回全局最优决策树。这个问题也可以由集成算法来解决，在随机森林中，特征和样本会在分枝过程中被随机采样。

- 有些概念很难学习，因为决策树不容易表达它们，例如 XOR，奇偶校验或多路复用器问题。

- 如果标签中的某些类占主导地位，决策树学习者会创建偏向主导类的树。因此，建议在拟合决策树之前平衡数据集。