信息论基础：

熵：

对随机事件的信息量求期望，得到随机变量X的熵：

$$H(X)=-\sum_{x\in X}P(x)\log P(x)$$

当对数底数是2时，单位是bit，当对数底数是e时，单位是nat(奈特)。同时，若$P(x)=0$,则定义$0\log 0=0$。由熵定义可知，随机变量的熵只依赖于X的分布，而与X的取值无关。
熵表示的是随机变量不确定性的度量。熵越大，随机变量的不确定性也就越大。

联合熵和条件熵：

设有随机变量(X,Y),其联合概率分布为：

$$P(X=x_i,Y=y_j)=p(x_i,y_j)=p_{ij},i=1,2,\cdots,n;j=1,2,\cdots,m$$

联合熵为$H(X,Y)=-\sum_{x,y}P(x,y)\log P(x,y)$
条件熵为$H(Y|X)=H(X,Y)-H(X)$。条件熵表示在已知随机变量X的条件下随机变量Y的不确定性。

$$\begin{aligned}
&H(Y|X)=H(X,Y)-H(X)\\
&=-\sum_{x,y}P(x,y)\log P(x,y)+\sum_{x}P(x)\log P(x)\\
&=-\sum_{x,y}P(x,y)\log P(x,y)+\sum_{x}\left(\sum_{y}P(x,y)\right)\log P(x)\\
&=-\sum_{x,y}P(x,y)\log P(x,y)+\sum_{x}\sum_{y}P(x,y)\log P(x)\\
&=-\sum_{x,y}P(x,y)\log\frac{P(x,y)}{P(x)}\\
&=-\sum_{x,y}P(x,y)\log P(y|x)\\
&=-\sum_{x}\sum_{y}P(x)P(y|x)\log P(y|x)\\
&=-\sum_{x}P(x)\sum_{y}P(y|x)\log P(y|x)\\
&=\sum_{x}P(x)\left(-\sum_{y}P(y|x)\log P(y|x)\right)\\
&=\sum_{x}P(x)H(Y|X=x)
\end{aligned}$$

信息增益：

信息增益是基于熵的概念的，熵表征的是信息的混乱程度，熵越大，数据集越混乱。信息熵表示的是得知特征X_j的信息而使所属分类的不确定性减少的程度，也就是能够最大程度地区分特征，一旦知道特征值属于某个区间，基本就能很大程度地提升其属于某个分类的概率。

对于一个训练集而言，令特征A对训练数据集D的信息增益为g(D,A)，那么这个训练集的信息熵就定义为集合D的经验熵H(D)，与特征A给定的情况下D的经验条件熵H(D|A)之差。

$$g(D,A)=H(D)-H(D|A)$$

假设数据集D有K种分类，特征A有n种取值可能。
其中数据集D的经验熵H(D)为

$$H(D)=-\sum_{k=1}^{K}P_k\log_2^{P_k}$$

其中P_k为集合D中的任一样本数据分类k的概率，或者说属于分类k的样本所占的比例。
经验条件熵H(D|A)为

$$H(D|A)=-\sum_{i=1}^{n}P_iH(D_i)$$

其中P_i为特征取值为第i个可取值的概率。D_i为特征A为第i个可取值的样本集合。

基尼不纯度：

前面介绍的信息增益主要是用于早期的决策树算法ID3算法，用作特征选择。但是使用信息增益作为特征选择容易出现特征可选值多时，信息增益越大的问题。直观地理解也可以知道了，当特征可选值多的时候，集合自然是不确定性越大了，所以基于熵计算出来的信息增益也就越大，所以C4.5改进了ID3，使用了信息增益比。

在C4.5的改进算法CART树中，使用的是基尼不纯度来衡量特征的重要性，主要用于解决信息增益和信息增益比存在大量对数运算，计算复杂度高的问题。基尼系数越小，则不纯度越低，特征越好。这和信息增益(比)是相反的。

对以某个特征为基准进行划分后的子集计算基尼不纯度，即计算随机放置的数据项出现于错误分类中的概率，以此来评判属性对分类的重要程度。下面是集合划分前的基尼不纯度：

$$Gini(D) = \sum_{k=1}^{K} P_k(1 - P_k) = 1 - {P_k}^2$$

其中P_k为任一样本点属于第k类的概率，也可以说成样本数据集中属于k类的样本的比例。
划分钱集合D的基尼指数为Gini(D)，以特征A为基准划分后集合D的基尼指数为：

$$Gini(D|A) = \sum |D_i||D|Gini(D_i)$$

其中$|D_i|$为特征A取第i个值时对应的样本个数。|D|为总样本个数。

决策树的不同分类算法：

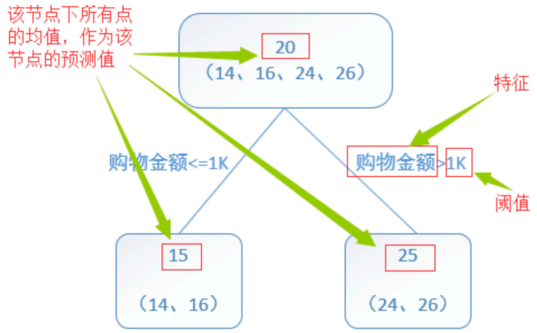
决策树是根据已知的若干条件，来对事件作出判断。从根节点到叶子节点，是将不同特征不断划分的过程，最后将类别分出。
ID3算法：
ID3算法将信息增益做贪心算法来划分算法，总是挑选是的信息增益最大的特征来划分数据，使得数据更加有序。检索哪个属性的分类能力更强，然后用拿个分类能力强的属性将数据分类，然后继续检索继续分类。最后分完之后就会是一个树的结构。分到最后数据会到达一定的纯度。ID3使用信息增益作为特征选择的度量。
1. 计算先验熵，没有接收到其他的属性值时的平均不确定性；
2. 计算后验墒，在接收到输出符号yi时关于信源的不确定性；
3. 条件熵，对后验熵在输出符号集Y中求期望，接收到全部的付好后对信源的不确定性；
4. 互信息，先验熵和条件熵的差；
C4.5:
用信息增益率来选择属性。在决策树构造过程中进行剪枝，因为某些具有很少元素的结点可能会使构造的决策树过适应（Overfitting），如果不考虑这些结点可能会更好。对非离散数据也能处理。能够对不完整数据进行处理。
CART分类树：
CART假设决策树是二叉树，内部结点特征的取值为“是”和“否”，左分支是取值为“是”的分支，右分支是取值为“否”的分支。这样的决策树等价于递归地二分每个特征，将输入空间即特征空间划分为有限个单元，并在这些单元上确定预测的概率分布，也就是在输入给定的条件下输出的条件概率分布。
CART算法由以下两步组成：
决策树生成：基于训练数据集生成决策树，生成的决策树要尽量大；
决策树剪枝：用验证数据集对已生成的树进行剪枝并选择最优子树，这时损失函数最小作为剪枝的标准。
CART决策树的生成就是递归地构建二叉决策树的过程。CART决策树既可以用于分类也可以用于回归。本文我们仅讨论用于分类的CART。对分类树而言，CART用Gini系数最小化准则来进行特征选择，生成二叉树。

回归树原理：

GBDT(Gradient Boosting Decision Tree) 又叫 MART (Multiple Additive Regression Tree)，是一种迭代的决策树算法，该算法由多棵决策树组成，所有树的结论累加起来做最

终答案。它在被提出之初就和SVM一起被认为是泛化能力较强的算法。GBDT中的树是回归树（不是分类树），GBDT用来做回归预测，调整后也可以用于分类。GBDT的思想使其具有天然优势可以发现多种有区分性的特征以及特征组合。业界中，Facebook使用其来自动发现有效的特征、特征组合，来作为LR模型中的特征，以提高 CTR预估（Click-Through Rate Prediction）的准确性；GBDT在淘宝的搜索及预测业务上也发挥了重要作用。

回归树总体流程类似于分类树，区别在于，回归树的每一个节点都会得一个预测值，以年龄为例，该预测值等于属于这个节点的所有人年龄的平均值。分枝时穷举每一个feature的每个阈值找最好的分割点，但衡量最好的标准不再是最大熵，而是最小化平方误差。也就是被预测出错的人数越多，错的越离谱，平方误差就越大，通过最小化平方误差能够找到最可靠的分枝依据。分枝直到每个叶子节点上人的年龄都唯一或者达到预设的终止条件(如叶子个数上限)，若最终叶子节点上人的年龄不唯一，则以该节点上所有人的平均年龄做为该叶子节点的预测年龄。（引用自一篇博客，详见参考文献3）



决策树防止过拟合手段：

产生过度拟合数据问题的原因有哪些?

原因1：样本问题

（1）样本里的噪音数据干扰过大，大到模型过分记住了噪音特征，反而忽略了真实的输入输出间的关系；

（2）样本抽取错误，包括样本数量太少，抽样方法错误，抽样时没有足够正确考虑业务场景或业务特点，等等导致抽出的样本数据不能有效足够代表业务逻辑或业务场景；

（3）建模时使用了样本中太多无关的输入变量。

原因2：构建决策树的方法问题

在决策树模型搭建中，我们使用的算法对于决策树的生长没有合理的限制和修剪的话，决策树的自由生长有可能每片叶子里只包含单纯的事件数据或非事件数据，可以想象，这种决策树当然可以完美匹配（拟合）训练数据，但是一旦应用到新的业务真实数据时，效果是一塌糊涂。

如何解决过度拟合数据问题的发生?

针对原因1的解决方法：合理、有效地抽样，用相对能够反映业务逻辑的训练集去产生决策树；

针对原因2的解决方法（主要）：剪枝：提前停止树的增长或者对已经生成的树按照一定的规则进行后剪枝。

模型评估：

自助法（bootstrap）：

训练集是对于原数据集的有放回抽样，如果原始数据集N，可以证明，大小为N的自助样本大约包含原数据63.2%的记录。当N充分大的时候，1-（1-1/N）^(N) 概率逼近 1-e^(-1)=0.632。抽样 b 次，产生 b 个bootstrap样本，则，总准确率为（accs为包含所有样本计算的准确率）：

$$acc_{boot} = \frac{1}{b}\sum_{i=1}^{b}(0.632 \times \varepsilon_i + 0.368 \times acc_s)$$

准确度的区间估计：

将分类问题看做二项分布，则有：
令 X 为模型正确分类，p 为准确率，X 服从均值 Np、方差 Np（1-p）的二项分布。acc=X/N为均值 p，方差 p（1-p）/N 的二项分布。
acc 的置信区间：

$$P\left(-Z_{\frac{\alpha}{2}} \leq \frac{acc-p}{\sqrt{p(1-p)/N}} \leq Z_{1-\frac{\alpha}{2}}\right) = 1-\alpha$$

$$P \in \frac{2 \times N \times acc + Z_{\frac{\alpha}{2}}^2 \pm Z_{\frac{\alpha}{2}}\sqrt{Z_{\frac{\alpha}{2}}^2 + 4 \times N \times acc - 4 \times N \times acc^2}}{2(N + Z_{\frac{\alpha}{2}}^2)}$$

sklearn参数详解，Python绘制决策树：

'''

scikit-learn中有两类决策树，它们均采用优化的CART决策树算法。

'''

```
from sklearn.tree import DecisionTreeRegressor
```

'''

回归决策树

'''

```
DecisionTreeRegressor(criterion="mse",
                      splitter="best",
                      max_depth=None,
                      min_samples_split=2,
                      min_samples_leaf=1,
                      min_weight_fraction_leaf=0.,
                      max_features=None,
                      random_state=None,
                      max_leaf_nodes=None,
                      min_impurity_decrease=0.,
                      min_impurity_split=None,
                      presort=False)
```

'''

参数含义：

1.criterion:string, optional (default="mse")

它指定了切分质量的评价准则。默认为'mse'(mean squared error)。

2.splitter:string, optional (default="best")

它指定了在每个节点切分的策略：

(1).splitter='best':表示选择最优的切分特征和切分点。

(2).splitter='random':表示随机切分。

3.max_depth:int or None, optional (default=None)

指定树的最大深度。如果为None，则表示树的深度不限，直到每个叶子都是纯净的，即叶节点中所有样本都属于同一个类别，或者叶子节点中包含小于min_samples_split个样本。

4.min_samples_split:int, float, optional (default=2)

整数或者浮点数，默认为2。它指定了分裂一个内部节点(非叶子节点)

需要的最小样本数。如果为浮点数(0到1之间)，最少样本分割数为ceil(min_samples_split * n_samples)

5.min_samples_leaf:int, float, optional (default=1)

整数或者浮点数，默认为1。它指定了每个叶子节点包含的最少样本数。

如果为浮点数(0到1之间)，每个叶子节点包含的最少样本数为ceil(min_samples_leaf * n_samples)

6.min_weight_fraction_leaf:float, optional (default=0.)

它指定了叶子节点中样本的最小权重系数。默认情况下样本有相同的权重。

7.max_feature:int, float, string or None, optional (default=None)

可以是整数，浮点数，字符串或者None。默认为None。

(1).如果是整数，则每次节点分裂只考虑max_feature个特征。

(2).如果是浮点数(0到1之间)，则每次分裂节点的时候只考虑int(max_features * n_features)个特征。

(3).如果是字符串'auto',max_features=n_features。

(4).如果是字符串'sqrt',max_features=sqrt(n_features)。

(5).如果是字符串'log2',max_features=log2(n_features)。

(6).如果是None，max_feature=n_feature。

8.random_state:int, RandomState instance or None, optional (default=None)

(1).如果为整数，则它指定了随机数生成器的种子。

(2).如果为RandomState实例，则指定了随机数生成器。

(3).如果为None，则使用默认的随机数生成器。

9.max_leaf_nodes:int or None, optional (default=None)

(1).如果为None，则叶子节点数量不限。

(2).如果不为None，则max_depth被忽略。

10.min_impurity_decrease:float, optional (default=0.)

如果节点的分裂导致不纯度的减少(分裂后样本比分裂前更加纯净)大于或等于min_impurity_decrease，则分裂该节点。

个人理解这个参数应该是针对分类问题时才有意义。这里的不纯度应该是指基尼指数。

回归生成树采用的是平方误差最小化策略。分类生成树采用的是基尼指数最小化策略。

加权不纯度的减少量计算公式为：

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

其中N是样本的总数，N_t是当前节点的样本数，N_t_L是分裂后左子节点的样本数，

N_t_R是分裂后右子节点的样本数。impurity指当前节点的基尼指数，right_impurity指

分裂后右子节点的基尼指数。left_impurity指分裂后左子节点的基尼指数。

11.min_impurity_split:float

树生长过程中早停止的阈值。如果当前节点的不纯度高于阈值，节点将分裂，否则它是叶子节点。

这个参数已经被弃用。用min_impurity_decrease代替了min_impurity_split。

12.presort： bool, optional (default=False)

指定是否需要提前排序数据从而加速寻找最优切分的过程。设置为True时，对于大数据集

会减慢总体的训练过程；但是对于一个小数据集或者设定了最大深度的情况下，会加速训练过程。

属性：
1.feature_importances_ ：array of shape = [n_features]
特征重要性。该值越高，该特征越重要。
特征的重要性为该特征导致的评价准则的（标准化的）总减少量。它也被称为基尼的重要性
2.max_feature_:int
max_features推断值。
3.n_features_： int
执行fit的时候，特征的数量。
4.n_outputs_ ：int
执行fit的时候，输出的数量。
5.tree_ ：底层的Tree对象。
Notes：
控制树大小的参数的默认值（例如``max_depth``, ``min_samples_leaf``等）导致完全成长和未剪枝的树，
这些树在某些数据集上可能表现很好。为减少内存消耗，应通过设置这些参数值来控制树的复杂度和大小。
方法：
1.fit(X,y):训练模型。
2.predict(X):预测。
'''

```
from sklearn.tree import DecisionTreeClassifier
'''
分类决策树
'''
DecisionTreeClassifier(criterion="gini",
                       splitter="best",
                       max_depth=None,
                       min_samples_split=2,
                       min_samples_leaf=1,
                       min_weight_fraction_leaf=0.,
                       max_features=None,
                       random_state=None,
                       max_leaf_nodes=None,
                       min_impurity_decrease=0.,
                       min_impurity_split=None,
                       class_weight=None,
                       presort=False)
'''
```

参数含义：
1.criterion:string, optional (default="gini")
(1).criterion='gini',分裂节点时评价准则是Gini指数。
(2).criterion='entropy',分裂节点时的评价指标是信息增益。
2.max_depth:int or None, optional (default=None)。指定树的最大深度。
如果为None，表示树的深度不限。直到所有的叶子节点都是纯净的，即叶子节点中所有的样本点都属于同一个类别。或者每个叶子节点包含的样本数小于min_samples_split。
3.splitter:string, optional (default="best")。指定分裂节点时的策略。
(1).splitter='best',表示选择最优的分裂策略。
(2).splitter='random',表示选择最好的随机切分策略。

4.min_samples_split:int, float, optional (default=2)。表示分裂一个内部节点需要的做少样本数。

(1).如果为整数，则min_samples_split就是最少样本数。

(2).如果为浮点数(0到1之间)，则每次分裂最少样本数为ceil(min_samples_split * n_samples)

5.min_samples_leaf: int, float, optional (default=1)。指定每个叶子节点需要的最少样本数。

(1).如果为整数，则min_samples_split就是最少样本数。

(2).如果为浮点数(0到1之间)，则每个叶子节点最少样本数为ceil(min_samples_leaf * n_samples)

6.min_weight_fraction_leaf:float, optional (default=0.)

指定叶子节点中样本的最小权重。

7.max_features:int, float, string or None, optional (default=None).

搜寻最佳划分的时候考虑的特征数量。

(1).如果为整数，每次分裂只考虑max_features个特征。

(2).如果为浮点数(0到1之间)，每次切分只考虑int(max_features * n_features)个特征。

(3).如果为'auto'或者'sqrt',则每次切分只考虑sqrt(n_features)个特征

(4).如果为'log2',则每次切分只考虑log2(n_features)个特征。

(5).如果为None,则每次切分考虑n_features个特征。

(6).如果已经考虑了max_features个特征，但还是没有找到一个有效的切分，那么还会继续寻找

下一个特征，直到找到一个有效的切分为止。

8.random_state:int, RandomState instance or None, optional (default=None)

(1).如果为整数，则它指定了随机数生成器的种子。

(2).如果为RandomState实例，则指定了随机数生成器。

(3).如果为None，则使用默认的随机数生成器。

9.max_leaf_nodes: int or None, optional (default=None)。指定了叶子节点的最大数量。

(1).如果为None,叶子节点数量不限。

(2).如果为整数，则max_depth被忽略。

10.min_impurity_decrease:float, optional (default=0.)

如果节点的分裂导致不纯度的减少(分裂后样本比分裂前更加纯净)大于或等于min_impurity_decrease，则分裂该节点。

加权不纯度的减少量计算公式为：

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

其中N是样本的总数，N_t是当前节点的样本数，N_t_L是分裂后左子节点的样本数，N_t_R是分裂后右子节点的样本数。impurity指当前节点的基尼指数，right_impurity指

分裂后右子节点的基尼指数。left_impurity指分裂后左子节点的基尼指数。

11.min_impurity_split:float

树生长过程中早停止的阈值。如果当前节点的不纯度高于阈值，节点将分裂，否则它是叶子节点。

这个参数已经被弃用。用min_impurity_decrease代替了min_impurity_split。

12.class_weight:dict, list of dicts, "balanced" or None, default=None

类别权重的形式为{class_label: weight}

(1).如果没有给出每个类别的权重，则每个类别的权重都为1。

(2).如果class_weight='balanced'，则分类的权重与样本中每个类别出现的频率成反

比。

计算公式为：n_samples / (n_classes * np.bincount(y))

(3).如果sample_weight提供了样本权重(由fit方法提供)，则这些权重都会乘以sample_weight。

13.presort:bool, optional (default=False)

指定是否需要提前排序数据从而加速训练中寻找最优切分的过程。设置为True时，对于大数据集

会减慢总体的训练过程；但是对于一个小数据集或者设定了最大深度的情况下，会加速训练过程。

属性:

1.classes_:array of shape = [n_classes] or a list of such arrays

类别的标签值。

2.feature_importances_ : array of shape = [n_features]

特征重要性。越高，特征越重要。

特征的重要性为该特征导致的评价准则的（标准化的）总减少量。它也被称为基尼的重要性

3.max_features_ : int

max_features的推断值。

4.n_classes_ : int or list

类别的数量

5.n_features_ : int

执行fit后，特征的数量

6.n_outputs_ : int

执行fit后，输出的数量

7.tree_ : Tree object

树对象，即底层的决策树。

方法:

1.fit(X,y):训练模型。

2.predict(X):预测

3.predict_log_poba(X):预测X为各个类别的概率对数值。

4.predict_proba(X):预测X为各个类别的概率值。

'''