分类模型：决策树

目录

[一、决策树的引入： 3](#_Toc484093617)

[二、首先从一个实例讲起： 4](#_Toc484093618)

[三、 什么是决策树/判定树（decision tree)? 9](#_Toc484093619)

[3.1 熵（entropy）概念： 11](#_Toc484093620)

[3.2 决策树归纳算法 （ID3） 13](#_Toc484093621)

[3.3信息熵和信息增益的计算示例（ID3关键步骤） 14](#_Toc484093622)

[3.4 决策树的其他优化算法： 16](#_Toc484093623)

[3.5 树剪枝叶 （避免overfitting过拟合) 17](#_Toc484093624)

[3.5.1 先剪枝 19](#_Toc484093625)

[3.5.2 后剪枝 19](#_Toc484093626)

[3.6 决策树算法的特点 22](#_Toc484093627)

[四、电商实例ID3算法详解 22](#_Toc484093628)

[1.ID3算法步骤演示 23](#_Toc484093629)

[2.按照ID3得到的决策树 24](#_Toc484093630)

[五、Spark模拟实现决策树 24](#_Toc484093631)

[1.Python代码实例： 2](#_Toc484093632)4

[2.Scala代码实例： 2](#_Toc484093633)4

[六、Scikit-Learn代码示例： 25](#_Toc484093634)

[1.Scikit-Learn库实现上述例子 36](#_Toc484093635)

[（1）scikit-learn实现待测数据的分类 3](#_Toc484093636)7

[（2）对上述示例的决策树过程进行分析 39](#_Toc484093637)

[2.Iris数据集的例子 43](#_Toc484093638)

[七、Python原生实现代码： 50](#_Toc484093639)

[ID3算法的Python实现 50](#_Toc484093640)

[1.ID3算法实现 50](#_Toc484093641)

[2.构建数据集 54](#_Toc484093642)

[3.训练的决策树的结构信息打印及字典转化为树状的显示 55](#_Toc484093643)

[4.持久化决策树 55](#_Toc484093644)

[5.测试ID3算法 56](#_Toc484093645)

[补充：C4.5算法Python实现 5](#_Toc484093646)6

[1.C4.5算法实现 5](#_Toc484093647)6

[2.构建数据集 5](#_Toc484093648)6

[3.训练决策树 5](#_Toc484093649)6

[4.测试C4.5算法 5](#_Toc484093650)6

[补充：基尼指数 56](#_Toc484093651)

# 决策树的引入：

有的同学可能在大学学习过一门课程叫《数据结构》，里面有一个重要的结构就是“树”，和现实生活中的树一样，树的主要由四部分[树根](https://baike.so.com/doc/6831821-7049020.html" \t "https://baike.so.com/doc/_blank)、[树干](https://baike.so.com/doc/6002536-6215513.html" \t "https://baike.so.com/doc/_blank)、[树枝](https://baike.so.com/doc/1378276-1456997.html" \t "https://baike.so.com/doc/_blank)、[树叶](https://baike.so.com/doc/5408467-5646454.html" \t "https://baike.so.com/doc/_blank)组成，今天的决策树也是一种树结构，大家学习的时候可以想象现实生活中的树来来理解。

**（强调）**决策树算法是一种监督学习算法，英文是Decision tree。

决策树思想的来源非常朴素，试想每个人的大脑都有类似于if-else这样的逻辑判断，这其中的if表示的是条件，if之后的then就是一种选择或决策。程序设计中的条件分支结构就是if-then结构，最早的决策树就是利用这类结构分割数据的一种分类学习方法。

比如：你母亲要给你介绍男朋友，是这么来对话的：

女儿：多大年纪了？

母亲：26。

女儿：长的帅不帅？

母亲：挺帅的。

女儿：收入高不？

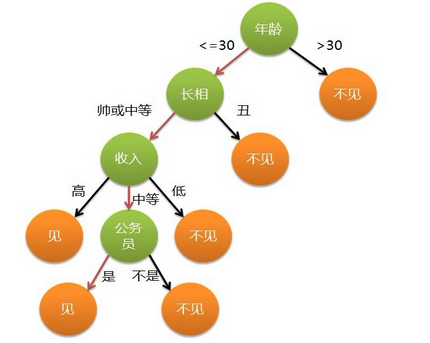
母亲：不算很高，中等情况。

女儿：是公务员不？

母亲：是，在税务局上班呢。

女儿：那好，我去见见。

于是你在脑袋里面就有了下面这张图：



作为女孩的你在决策过程就是典型的分类树决策。相当于通过年龄、长相、收入和是否公务员对将男人分为两个类别：见和不见。

**有了上面这个认识，我们就通过之前比较熟悉电商例子开始了解决策树算法。**

# 2.首先从一个实例讲起：

在商业的数据挖掘中，不同的消费行为顾客特征的提炼和表述极为重要。

我们模拟前期大家做过的电商平台项目中，模拟顾客及其消费行为数据共1024条，包括用户是否购买某种产品（0购买，1不购买），年龄（青年0，中年1，老年2），收入（高0，中1，低2），学生（是1，否0），信誉（良0，优1）。

我们模拟收集了如表1-1的部分用户购买数据，建立了一张统一的调查表，统计几个月的销售数据。我们这里模拟通过从电商平台中对下表中潜在的客户进行分析，并根据得到的一些特征用于销售人员制定销售策略等工作。

总结为两个问题：

(1) 如何对客户进行分类？

(2) 如何根据分类的依据，给出销售人员的指导意见？

**补充和解释：**

分析一个机器学习问题：

1.首先搞清楚数据集的内容

2.数据集：行--样本(15)、列---特征个数(4)

3.特征：(4)

年龄---青年、中年、老年

收入----高、中、低

学生----是 or 否

信誉----优 良

4.类标签：(1)

最后一列：买 or 不买 （二分类问题）

5.训练集(训练模型) + 测试集(测试模型的优劣)

问题：

1.如何对客户进行分类？

答：根据数据集中收集的用户的特征信息

2.如何根据分类依据，给出销售指导意见呢？

答：给出这样的销售意见：

中年人通常会无条件购买

青年人中如果是学生一般会购买

老年人中信誉好的常常会购买

表 1-1 用户购买统计表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 统计用户计数 | 年龄 | 收入 | 学生 | 信誉 | 是否购买 |
| 64 | 青 | 高 | 否 | 良 | 不买 |
| 64 | 青 | 高 | 否 | 优 | 不买 |
| 128 | 中 | 高 | 否 | 良 | 买 |
| 60 | 老 | 中 | 否 | 良 | 买 |
| 64 | 老 | 低 | 是 | 良 | 买 |
| 64 | 老 | 低 | 是 | 优 | 不买 |
| 64 | 中 | 低 | 是 | 优 | 买 |
| 128 | 青 | 中 | 否 | 良 | 不买 |
| 64 | 青 | 低 | 是 | 良 | 买 |
| 132 | 老 | 中 | 是 | 良 | 买 |
| 64 | 青 | 中 | 是 | 优 | 买 |
| 32 | 中 | 中 | 否 | 优 | 买 |
| 32 | 中 | 高 | 是 | 良 | 买 |
| 64 | 老 | 中 | 否 | 优 | 不买 |

1. 对表格数据的分析

从第一列中可以看到这张表格，共15行，每行表示列特征不同取值的统计人数。**第二列是年龄特征**，取值有三个：老、中、青。**第三列是收入**：取值是高、中、低。**第四列是学生**，取值有是获否。**第五列为信誉**，取值有两个，为优和良。最后一列为**销售的结果**，也就是分类标签，取值为买或不买。

1. 问题分析：

那么对于任意给定特征值的一个客户（也就是测试样例），算法需要帮助公司将这位客户归类，即预测这位客户是属于“购买”商品的一类还是“不购买”商品的一类并且给出判断依据。

1. 首先，我们根据示例**手工**实现一颗决策树。

**分析规则：（制定规则）**

a.既然是一棵树，肯定是由根节点、叶子结点和内部节点组成。

b.(根节点选取)如果从一颗空决策树开始，任意选择第一个特征就是根节点。

c.(叶子结点的判断)如果我们按照条件进行划分，若划分的到一个子集为空或子集中所有样本已经归为同一类的别标签，则该子集就是叶子结点，否则这些子集就对应于决策树中的内部节点。

d.(内部节点需要继续划分)如果是内部节点，就需要选择一个新的类别标签继续对该子集进行划分，直到所有子集为叶子节点。

**根据规则进行划分：** 根据上述树的规则我们对示例数据集进行划分。

1. 我们选年龄为根节点，该特征值取值为：老、中、青。根据年龄特征将所有样本进行划分，并作为决策树的第一层。

（1）年龄 = 青，是否购买：不买、买

（2）年龄 = 中，是否购买：买

（3）年龄=老，是否购买：不买、买

也就是当年龄为中年，是否购买都一致地变为买，此时的中年就成为决策树的叶子节点。而当年龄为青年和老年时候，是否购买有两个选择，可以继续分解。

如下表1-2：按照年龄分开数据集：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 统计用户计数 | 年龄 | 收入 | 学生 | 信誉 | 是否购买 |
| 64 | 青 | 高 | 否 | 良 | 不买 |
| 64 | 青 | 高 | 否 | 优 | 不买 |
| 128 | 青 | 中 | 否 | 良 | 不买 |
| 64 | 青 | 低 | 是 | 良 | 买 |
| 64 | 青 | 中 | 是 | 优 | 买 |
| 128 | 中 | 高 | 否 | 良 | 买 |
| 64 | 中 | 低 | 是 | 优 | 买 |
| 32 | 中 | 中 | 否 | 优 | 买 |
| 32 | 中 | 高 | 是 | 良 | 买 |
| 60 | 老 | 中 | 否 | 良 | 买 |
| 64 | 老 | 低 | 是 | 良 | 买 |
| 64 | 老 | 低 | 是 | 优 | 不买 |
| 132 | 老 | 中 | 是 | 良 | 买 |
| 64 | 老 | 中 | 否 | 优 | 不买 |

1. 接着，年龄为“青年”、“老年”需要接着划分。对数据集中“青年”选项的行数据继续选择第二个特征，如：收入，根据收入得到如下表1-3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 统计用户计数 | 年龄 | 收入 | 学生 | 信誉 | 是否购买 |
| 64 | 青 | 高 | 否 | 良 | 不买 |
| 64 | 青 | 高 | 否 | 优 | 不买 |
| 128 | 青 | 中 | 否 | 良 | 不买 |
| 64 | 青 | 中 | 是 | 优 | 买 |
| 64 | 青 | 低 | 是 | 良 | 买 |

1. 根据上表分析，高收入和低收入的特征值只有一个类别标签---买，将其作为叶子节点，然后继续划分中等以下收入的下一个特征---学生，如下表1-4：根据是学生将不会购买此商品，不是学生将会购买此商品。到此，年龄为“青年”的全部归类。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 统计用户计数 | 年龄 | 收入 | 学生 | 信誉 | 是否购买 |
| 128 | 青 | 中 | 否 | 良 | 不买 |
| 64 | 青 | 中 | 是 | 优 | 买 |

1. 青年划分完之后继续标签为老年的划分，如我们选择划分的顺序为信誉---收入---学生----计数，如此，当选择信誉作为下一级划分数据集的特征时，一次性就将信誉为优的类别标签划分为不买，信誉为良的类别标签划分为买。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 统计用户计数 | 年龄 | 收入 | 学生 | 信誉 | 是否购买 |
| 60 | 老 | 中 | 否 | 良 | 买 |
| 64 | 老 | 低 | 是 | 良 | 买 |
| 64 | 老 | 中 | 是 | 良 | 买 |
| 132 | 老 | 低 | 是 | 优 | 不买 |
| 64 | 老 | 中 | 否 | 优 | 不买 |

根据以上几部，我们可以得到一颗完整的决策树，我们为了判断方便将所有购买商品的叶子节点放在树的右侧，不购买商品的叶子结点放在树的左侧。

a.从定性的角度画出决策树：



b.从定量的角度画出决策树：（根据表格进行量化）

为什么需要从定量角度来分析呢？

答：这样会更精确的分析用户的特征信息，给出销售人员更准确的数据信息。



# 构建决策树的三个步骤

构建决策树包括三个步骤：

* 特征选择：选取有较强分类能力的特征。
* 决策树生成：典型的算法有ID3和C4.5，它们生成决策树过程相似，ID3是采用信息增益作为特征选择度量，而C4.5采用信息增益比率。
* 决策树剪枝：剪枝原因是决策树生成算法生成的树对训练数据的预测很准确，但是对于未知数据分类很差，这就产生了过拟合的现象。

注意：决策树的构建直到没有特征可选或者信息增益很小时停止，这就导致了构建的决策树模型过于复杂，而复杂的模型是在训练数据上建立的，所以对于测试集往往造成分类不准确，这就是过拟合。

发生过拟合是由于决策树太复杂，解决办法是控制模型的复杂度，对于决策树来说就是简化模型，称为“剪枝”。

决策树剪枝的过程就是从已生成的决策树上裁掉一些子树或者叶节点。剪枝的目标是通过极小化决策树的整体**损失函数**来实现的。

下面我们从决策树目标入手，讲一下如何构建决策树模型。

决策树的目标：给定训练集，其中为输入实例，n为样本个数（），为类标记，i=1,2.....N;N为样本容量。构建决策树的目标是，根据给定的训练数据集学习一个决策树模型。

构建决策树通常是递归地选择最优特征，并根据该特征对训练数据进行分割，步骤如下：

1. 构建根节点，所有的样本都位于根节点。
2. 选择一个最优的特征。通过该特征将训练数据集分割成子集，确保各个子集有最好的分类，但要考虑以下两种情况：

* 若子集能够被“较好地”分类，则构建叶节点，并将子集划分到对应的叶节点中去。
* 若某个子集不能被“较好地”分类，则对该子集继续划分。

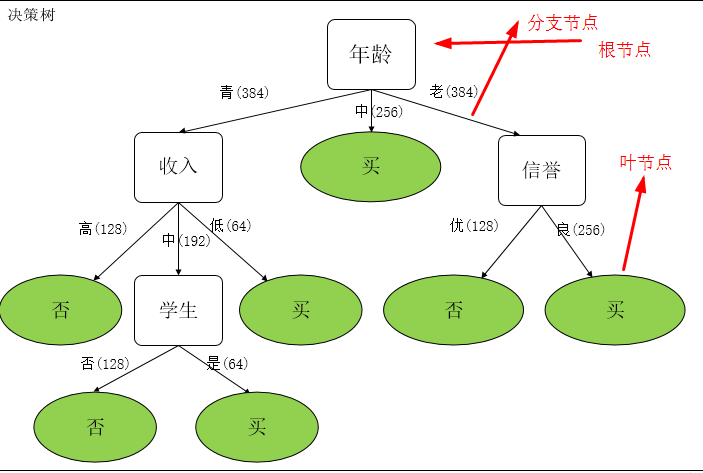
1. 递归直到所有训练样本都被很好的分类，或没有合适的特征为止。对于是否较好的分类，我们借助“信息增益”、“信息增益率”、“基尼系数”、“分类错误率”等指标衡量。

通过上述步骤生成的决策树对训练样本有很好的分类能力，但是我们需要的是对未知样本的分类能力。因此通常需要对已生成的决策树进行剪枝，从而使得决策树有很好的泛化能力。剪枝的过程就是去掉过于细分的叶节点，从而提高泛化能力。

# 4.决策树的基本概念和算法

## 4.0什么是决策树/判定树（decision tree)?

1.决策树是一个类似于流程图的树结构：其中，每个内部结点表示一个特征或属性，而每个树叶结点代表一个分类。树的最顶层是根结点。使用决策树分类时就是将实例分配到叶节点的类中。该叶节点所属的类就是该节点的分类。（通过下图理解）



2.  决策树是机器学习中分类方法中的一个重要算法

3.  构造决策树的基本算法

（1）ID3算法：使用信息增益进行特征选择

（2）C4.5算法：使用信息增益率进行特征选择，克服了信息增益选择特征的时候偏向于特征个数较多的不足。

（3）CART算法：CART全称为分类回归树，既可以用于分类，也可以用于预测。利用CART构建回归树用到树的**剪枝**技术，用于防止树的过拟合。

通过以下表格，我们通过观察并得到了一颗决策树，那么我们如何根据ID3算法得到我们优化过的决策树呢？这就需要**首先引入熵的概念**。



表1 用户购买统计表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 统计用户计数 | 年龄 | 收入 | 学生 | 信誉 | 是否购买 |
| 64 | 青 | 高 | 否 | 良 | 不买 |
| 64 | 青 | 高 | 否 | 优 | 不买 |
| 128 | 中 | 高 | 否 | 良 | 买 |
| 60 | 老 | 中 | 否 | 良 | 买 |
| 64 | 老 | 低 | 是 | 良 | 买 |
| 64 | 老 | 低 | 是 | 优 | 不买 |
| 64 | 中 | 低 | 是 | 优 | 买 |
| 128 | 青 | 中 | 否 | 良 | 不买 |
| 64 | 青 | 低 | 是 | 良 | 买 |
| 132 | 老 | 中 | 是 | 良 | 买 |
| 64 | 青 | 中 | 是 | 优 | 买 |
| 32 | 中 | 中 | 否 | 优 | 买 |
| 32 | 中 | 高 | 是 | 良 | 买 |
| 64 | 老 | 中 | 否 | 优 | 不买 |

## 4.1 熵（entropy）概念：

**(1)信息如何度量？**

  1948年，香农提出了 “信息熵(entropy)”的概念

一条信息的信息量大小和它的不确定性有直接的关系，要搞清楚一件非常非常不确定的事情，或者是我们一无所知的事情，需要了解大量信息==>（也就是说）**信息量的度量就等于不确定性的多少**

  （使用）比特(bit)来衡量信息的多少

**例子：**对一个有32支球队的世界杯争夺冠军，熵为？

其中p1,p2……p32分别是32支球队争冠的概率。如果当32支球队夺冠概率相同时，对应的熵是最大的，等于5bit。（分析如何得出5这个数值的。）

注意：实质上每个队夺冠的几率不是相等的。当概率不相等的时候算出的熵都比5小。

**（2）为什么要有“熵”的概念？为什么是这样一个公式？**

既然决策树有根节点、内部节点和叶子节点之分，那么究竟那个属性做根节点，那个属性又做下一级分类的分支节点(内部节点)?

香农在他的<<信息论>>中借用德国物理学家发明的“熵”的概念提出了“信息熵”的概念。在物理学中“熵”通常是指**物体能量的分布更加均匀**，而香农首先定义了“信息”的概念：信息就是对不确定性的消除。因此，**“信息熵”就表示事物不确定性的度量标准，可以根据数学中的概率计算，出现的概率就大，出现的机会就多，不确定性就小（信息熵小）。**

通常**多个独立事件所产生的不确定性等于各自不确定性之和**，也就是每个事件的发生与不发生是相互独立的，这在数学上称为可加性，而满足这样可加性的函数就是我们在高中阶段所学习过的对数函数(log函数)。由此得到第一个**不确定性函数**公式：

  如果对一个随机事件有n种取值：X1,X2,X3…，对应的概率p1,p2…..pn,则该事件的不确定性应该是单个取值的不确定性的期望E，也叫作信息熵，即：

上述若对数的底数取值为2，就是我们平常所说的信息**单位bit(比特)**。

**#总结#** 变量的不确定性越大，熵也就越大。熵越小，信息的纯度越高。

有了以上基础之后，我们就可以学习ID3算法，从而构建我们的决策树，从而达到分类目的。

## 4.2决策树归纳算法 （ID3）

* 1970-1980， J.Ross. Quinlan, ID3算法
* 目的：选择属性判断结点
* 引入“信息增益”：信息获取量(Information Gain)：

Gain(A) = Info(D)-Info\_A(D)

注：整个数据集信息熵与当前节点（划分节点）信息熵的差

* 通过某一个节点来作为分支节点来分类获取了多少信息

**（1）信息增益概念的含义和本质？**

**含义：**划分数据集前后信息发生的变化。

**本质**上是衡量给定属性能否更完美、没有误差地区分开训练数据集的能力。

我们使用信息增益来确定决策树分支的划分依据，它也是决策树分支上整个数据集信息熵与当前节点（划分节点）信息熵的差值，通常用Gain(A)表示：

也就是当前节点信息熵E(A)越小，不确定性也越小，整体数据集的信息熵（I(parent)）一般是保持不变的，从而当前节点的信息增益就越大，因此通常选择信息熵小的，信息增益大的值作为根节点。

**（2）ID3算法简介：**

ID3算法起源于CLS概念学习系统，以信息熵的下降速度为选取测试属性的标准，即在每个节点选取尚未被用来划分的具有最高信息增益的属性作为划分标准，继续这个过程，直到生成决策树能完美分类训练样例。

**输入：**样本的集合S，属性集合A

**输出：**ID3决策树

1. 如果所有种类的属性都处理完毕，则返回。否则，执行步骤B.
2. 计算出**信息增益最大**的属性a，将该属性作为一个节点。如果仅凭借a就可以对样本分类，则返回。否则执行步骤C。
3. 对属性a的每个可能取值v执行以下操作：

1.将v的样本作为s的一个子集Sv

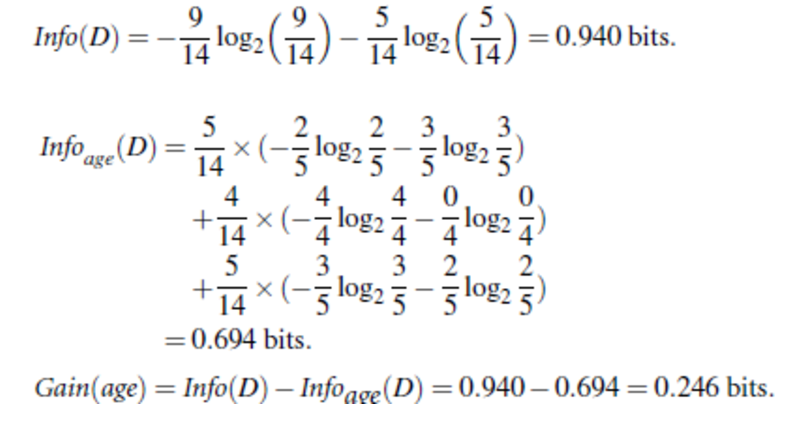
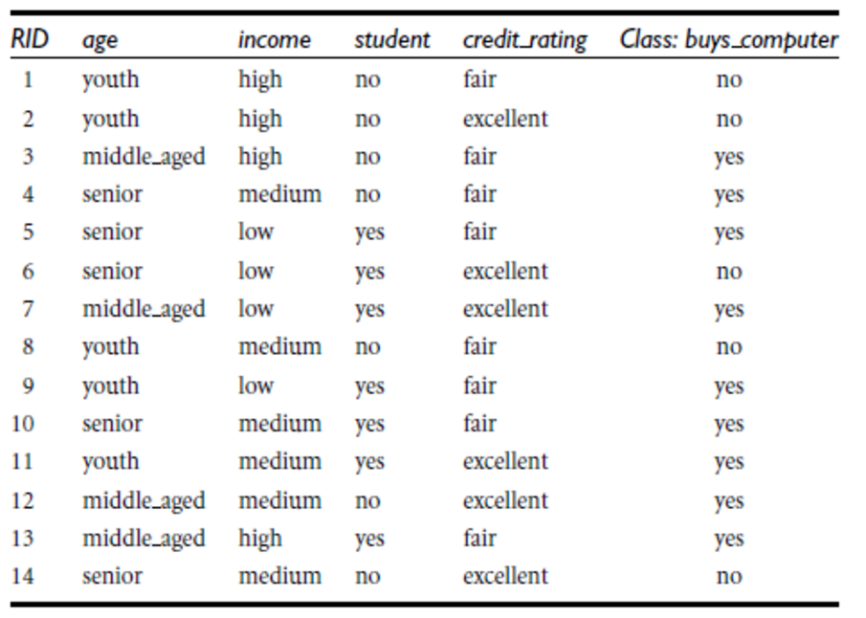
2.生成剩余属性集合AT=A-{a}

3.以样本集合Sv和属性集合AT为输入，递归执行ID3算法。

**了解了信息熵和熵的计算之后，我们通过简单例子计算这两个重要的变量。**

## 4.3信息熵和信息增益的计算示例（ID3关键步骤）

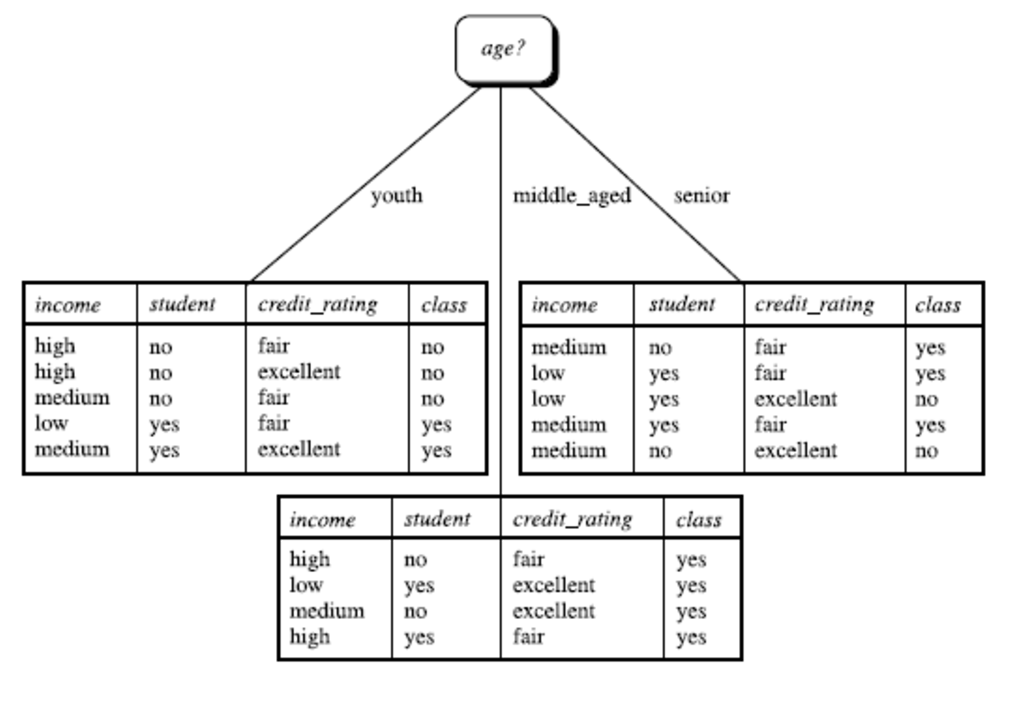
有了以上理论基础我们可以接下来通过一个小例子完整计算信息熵和信息增益，从而通过ID3算法构造决策树。



注：整个数据集信息熵与当前节点（划分节点）信息熵的差，就是信息增益

  类似，Gain(income) = 0.029, Gain(student) = 0.151, Gain(credit\_rating)=0.048

  所以，选择age作为第一个根节点



重复上述过程。

**算法步骤：(和前面的算法步骤理解其一即可)**

* 树以代表训练样本的单个结点开始（步骤1）。
* 如果样本都在同一个类，则该结点成为树叶，并用该类标号（步骤2 和3）。
* 否则，算法使用称为**信息增益**的基于熵的度量作为启发信息，选择能够最好地将样本分类的属性（步骤6）。该属性成为该结点的“测试”或“判定”属性（步骤7）。在算法的该版本中，
* 所有的属性都是分类的，即离散值。连续属性必须离散化。
* 对测试属性的每个已知的值，创建一个分枝，并据此划分样本（步骤8-10）。
* 算法使用同样的过程，递归地形成每个划分上的样本判定树。一旦一个属性出现在一个结点上，就不必在该结点的任何后代上考虑它（步骤13）。
* **递归划分步骤仅当下列条件之一成立停止：**

(a) 给定结点的所有样本属于同一类（步骤2 和3）。

(b) 没有剩余属性可以用来进一步划分样本（步骤4）。在此情况下，使用多数表决（步骤5）。

这涉及将给定的结点转换成树叶，并用样本中的多数所在的类标记它。替换了可以存放结点样本的类分布。

(c) 分枝：test\_attribute = a ，没有样本（步骤11）。在这种情况下，以 samples 中的多数类创建一个树叶（步骤12）

**定义一个最大不纯度，如果最大不纯度下降小于a时，就将该节点看做是叶子节点。(或可理解为纯度达到阈值a)**

**通过这个小例子大家了解了通过ID3算法构建树的过程，我们在后面还会利用熵和信息熵计算我们电商的例子，接下来我们了解下除了ID3算法，还有哪几种算法可以来构建决策树。**

## 4.4决策树的其他优化算法：

**（1）C4.5: 1993年 Quinlan**

C4.5算法

C4.5和ID3（1986年**Quinlan提出的**）算法类似，只有在以下几个点作了改进。

1. 用**信息增益率**来选择属性，克服用信息增益选择时候偏向选择取值多的属性的不足。（了解）

**特征A对数据集D的**信息增益：Gain(A) = Info(D)-Info\_A(D)

信息增益率：Gainr(A) =Gain(A) /**H(A)** ：其中H(A)为A的熵

**理解：**

由于在分类问题困难时候，也就是在训练数据集**经验熵**(**总体的Info(D)值**)大的时候，信息增益值会偏大(Info(D)值偏大)，如果训练数据集**经验熵**小的时候，信息增益会偏小，这时候也可以通过信息增益率来矫正这一问题。

**经验熵(就是前面提到的信息熵，也就是信息量的大小问题)**

2.在树构造过程中进行剪支。

3.能够完成连续属性的离散化处理。

4.能够对不完整数据进行处理。

**（2）CART**

Classification and Regression Trees (**CART)**: (L. Breiman, J. Friedman, R. Olshen, C. Stone)

**共同点：**都是贪心算法，自上而下(Top-down approach)

**区别：**属性选择度量方法不同

C4.5 （gain ratio)，CART(gini index)，ID3 (Information Gain)

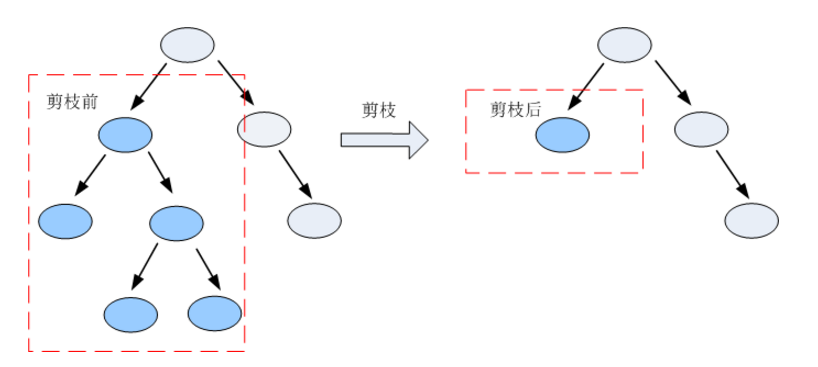
**贪心算法：**顾名思义，贪心算法总是**作出在当前看来最好的选择**。也就是说贪心算法并不从整体最优考虑，它所作出的选择只是在某种意义上的局部最优选择。

**无论哪种方法构建树，因为树的构建过程是递归形式的，所以有可能出现树的过拟合情况，因此，我们对树过拟合情况有一种“树剪枝”技术，避免树由于过度生长而出现的过拟合现象。接下来我们讲解树的两种剪枝技术。**

## 4.5树剪枝叶(避免overfitting过拟合)

**问题1：什么是剪枝？**

剪枝是指将一颗子树的子节点全部删掉，利用叶子节点替换子树(实质上是后剪枝技术)，也可以（假定当前对以root为根的子树进行剪枝）只保留根节点本身而删除所有的叶子，以下图为例：



**问题2：为什么要进行树的剪枝？**

决策树是充分考虑了所有的数据点而生成的复杂树，有可能出现过拟合的情况，决策树越复杂，过拟合的程度会越高。

考虑极端的情况：如果我们令所有的叶子节点都只含有一个数据点，那么我们能够保证所有的训练数据都能准确分类，但是很有可能得到高的预测误差，原因是将训练数据中所有的噪声数据都”准确划分”了，强化了噪声数据的作用。

而剪枝修剪分裂前后分类误差相差不大的子树，能够降低决策树的复杂度，降低过拟合出现的概率。

**注意：（解释）**

因为决策树的构建过程是一个递归的过层，所以必须确定停止条件，否则过程将不会停止，树会不停生长。通过我们前面的例子我们可以当一个节点下面的所有记录都属于同一类，或者当所有记录属性都具有相同的值时停止，但是这样往往会使得树的节点过多，导致过度拟合的问题。

过度拟合是指直接生成的完全决策树对训练样本的特征描述的“过于精确”，无法实现对新样本进行合理的分许，所以这种情况我们构建的树不是一颗最佳的决策树。

所以，为了避免过拟合，我们引入剪枝技术。

除了剪枝技术我们还有一种解决方法：当前结点中的记录数低于一个最小阈值就停止分裂，采用多数表决的方法决定叶节点的分类。

**问题3：怎样剪枝？**

两种方案：先剪枝和后剪枝

 先剪枝说白了就是提前结束决策树的增长，跟上述决策树停止生长的方法一样。

后剪枝是指在决策树生长完成之后再进行剪枝的过程。

接下来我们深入理解下这两种剪枝技术。

### 3.5.1 先剪枝（预剪枝）

先剪枝是对决策树停止标准的修改。

在ID3算法中，节点的分割一直到节点中的实例属于同一类别的时候才停止。对于包含较少实例的节点，很有可能被分割为单一实例的节点。为了避免这种情况，我们给出一个阈值，当一个节点分割导致的**最大的不纯度下降小于a时**，就把该节点看作是叶子结点。该方法选择的阈值对决策树的构建有很大的影响。

（1）当阈值a选择的过大的时候，节点的不纯度依然很高就停止了分裂。此时的树由于生长不足，导致决策树过小，分类的错误率过高。因此需要调整a参数的合理范围之内的值。

（2）当阈值a选择的过小的时候，比如a接近0，节点的分割过程近似于原始的分割过程。

由此可见，预剪枝方法虽然简单，但在实际应用中对a的选择存在很大的主观性。精确的给出a的值也是相当困难。

### 3.5.2 后剪枝

后剪枝是从一个充分生长的树中，按照自低向上的方式修剪掉多余的分支，有两种方法：

（1）用新的叶子结点替换子树，该叶子结点的类标号由子树记录中的多数类决定。

（2）用子树中最常用的分支替代子树。

通常计算前后预期分类错误率，如果修剪导致预期分类错误率变大，则放弃修剪，保留相应的各个分支，否则就将相应的节点分支修剪删除。

通常采用后剪枝技术是最小的错误剪枝(MEP)技术，即在产生一系列的经过修剪后的决策树候选之后，**利用一个独立的测试数据集，对这些经过修剪之后的决策树的分类准确性进行评价**，保留下预期分类错误率最小的（修剪后）决策树。

除了最小错误剪枝技术外，还有悲观错误剪枝(MEP)和代价复杂度剪枝(CCP)

### 3.5.3剪枝技术对比

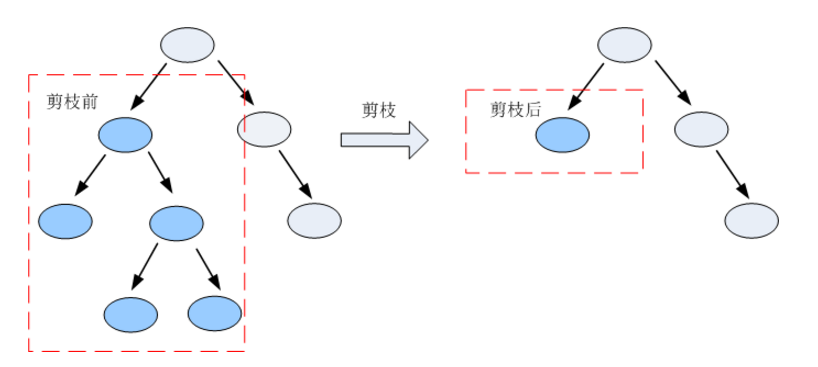
预剪枝使得决策树的很多分支没有展开，这不仅仅降低了过拟合的风险，还显著减少了决策树的训练时间开销和测试时间开销。但另一方面，有些分支的当前划分虽不能提升泛化性能，甚至会导致泛化性能降低，但在其基础上进行的后续划分却有可能导致性能的显著提高。同时，预剪枝决策树也带来了欠拟合的风险。

后剪枝的决策树通常比预剪枝的决策树保留了更多的分支。一般情况下，后剪枝决策树的欠拟合风险很小，泛化性能往往优于预剪枝决策树。但后剪枝过程是在生成完全决策树之后进行的，并且要**自底向上**地对树中所有非叶子节点进行逐一考察，因此在训练时间开销比未剪枝的决策树和预剪枝的决策树都要大得多。

**接下来对决策树算法的特点进行总结：**

### 3.5.4剪枝算法的数学原理（了解）

剪枝总体思路：



由完全树T0开始，剪枝部分节点得到T1，再次剪枝部分节点得到T2...直到仅剩树根的树Tk；在验证数据集上对这K个树分别评价，选择损失函数最小的树的树Ta。

#### 3.5.4.1剪枝系数

根据原损失函数：



叶节点越多，决策树越复杂，损失越大，修正：



当时，未剪枝的决策树损失最小（全树）；

当时，单根结点的决策树损失最小（单根）

**参考回归问题损失函数定义理解：**



现在考虑以r为根的子树剪枝，且剪枝后只保留r本身而删掉所有的叶子：

剪枝后的损失函数：

剪枝前的损失函数：

剪枝前后要求损失函数下降或至少相等，这里取剪枝前后相等得到剪枝系数a：



#### 3.5.4.2剪枝算法

对于给定的决策树T0

* 计算所有内部节点的剪枝系数a
* (a越大，表示单根，就会越安全，a越小表示未剪枝)
* 查找最小剪枝系数a的结点，剪枝得决策树Tk
* 重复以上步骤，直到决策树TK只有1个结点
* 得到决策树序列T0、T1、T2、T3..............Tk
* 使用验证样本集选择最优子树，可以使用损失函数作为评价最优子树的标准



## 4.6 决策树算法的特点

（1） 决策树的优点：

* 直观，便于理解，小规模数据集有效
* 执行效率高，执行只需要一次构建，可反复使用

（2）决策树的缺点：

* 处理连续变量不好，较难预测连续字段
* 类别较多时，错误增加的比较快
* 对于时间序列数据需要做很多的预处理
* 可规模性一般
* 实际分类的时候只能根据一个字段进行

## 4.7 决策树算法处理连续值

连续值的处理：由于连续属性的可取值数目不在有限，因此，不能直接根据连续属性的取值对结点进行划分。此时，我们采用连续值离散化的技术，最简单的是采用二分法（将样本的属性取值从大到小排序，找一个划分点将样本集分成两个子集，大于划分点的集合是决策树的一个分支，小于划分点的是决策树的一个分支）对连续属性进行处理，这也是C4.5算法采用的策略。

# 5.电商实例ID3算法详解

表1-2：按照年龄分开数据集：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 统计用户计数 | 年龄 | 收入 | 学生 | 信誉 | 是否购买 |
| 64 | 青 | 高 | 否 | 良 | 不买 |
| 64 | 青 | 高 | 否 | 优 | 不买 |
| 128 | 青 | 中 | 否 | 良 | 不买 |
| 64 | 青 | 低 | 是 | 良 | 买 |
| 64 | 青 | 中 | 是 | 优 | 买 |
| 128 | 中 | 高 | 否 | 良 | 买 |
| 64 | 中 | 低 | 是 | 优 | 买 |
| 32 | 中 | 中 | 否 | 优 | 买 |
| 32 | 中 | 高 | 是 | 良 | 买 |
| 60 | 老 | 中 | 否 | 良 | 买 |
| 64 | 老 | 低 | 是 | 良 | 买 |
| 64 | 老 | 低 | 是 | 优 | 不买 |
| 132 | 老 | 中 | 是 | 良 | 买 |
| 64 | 老 | 中 | 否 | 优 | 不买 |

## 5.1.ID3算法步骤演示

有了上面的基础之后，通过最上面的例子实现ID3算法的生成过程。

1. 计算对给定样本分类所需的信息熵

类别标签S被分为两类：买or不买。其中：

S1（买）=640；S2（不买）=384；S=S1+S2=1024;

S1的概率p1=640/1024=0.625; S2的概率p2=384/1024=0.375

**I(S1,S2)=I(640,384)=-p1log(p1) -p2log(p2)=0.9544**

1. 计算每个特征的信息熵及信息增益
2. 计算“年龄”特征的信息熵

年龄分为三组：青年（0）、中年（1）、老年（2）

* 1. 青年占总样本的概率

**P(0)**=384/1024=0.375

S1(买)=128 p1=128/384

S2(不买)=256 p2=256/384

**I1(S1,S2)=I(128,256)=-p1log(p1) -p2log(p2)=0.9183**

* 1. 中年占总样本的概率

**P(1)**=256/1024=0.25

S1(买)=256 p1=256/256=1

S2(不买)=0 p2=0/384=0

**I2(S1,S2)=I(256,0)=-p1log(p1) -p2log(p2)=0**

* 1. 老年占总样本的概率

**P(2)**=384/1024=0.375

S1(买)=257 p1=257/384

S2(不买)=127 p2=127/384

**I3(S1,S2)=I(257,127)=-p1log(p1) -p2log(p2)=0.9157**

则年龄的信息熵为：

E（年龄）= **P(0)\* I1(S1,S2)+ P(1)\* I2(S1,S2)+ P(2)\* I3(S1,S2)**

**=0.375\*0.9183+0.25\*0+0375\*0.9157 =0.6877**

则年龄的信息增益为：

G（年龄）=**0.9544-0.6877=0.2667**

1. 计算“学生”特征的信息熵

E（学生）=0.7811

G（学生）=0.9544-0.7811=0.1733

1. 计算“收入”“特征的熵

E（收入）=0.9361

G（收入）=0.9544-0.9361=0.0183

1. 计算“信誉”“特征的熵

E（信誉）=0.9048

G（信誉）=0.9544-0.9048=0.0496

1. 从所有列的特征中选出信息增益最大的那个作为根节点或内部节点

根据上述信息增益的大小，选定年龄列G=0.2667来划分数据集。

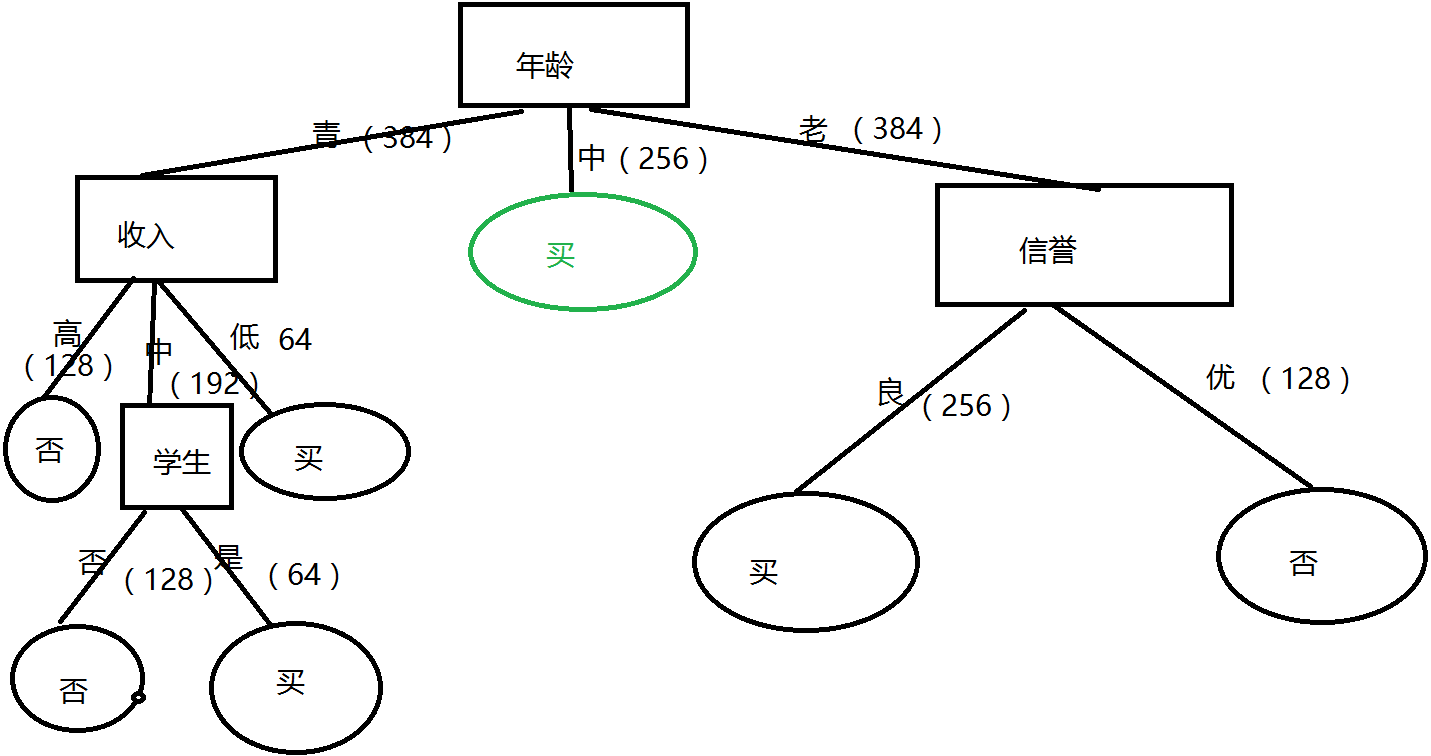
1. 根据节点的不同取值将数据集拆分为若干子集，删除当前的特征列，在计算剩余列的特征信息熵。如果有信息增益，就重复第二步直至划分结束。

* 首次划分后，青年和老年内含有多个标签，所以可以继续划分，中年选项就剩下一个标签就称作为叶子结点。

1. 划分结束标志为：子集中只有一个类别标签，停止划分。

**根据信息增益，选择出根节点，依次划分得到一颗决策树，大家可以对比下这颗决策树和之前的决策树的差别在哪里？**

**之前的决策树如下：**



## 5.2.按照ID3得到的决策树

按照上述结果产生一个决策树如下图：

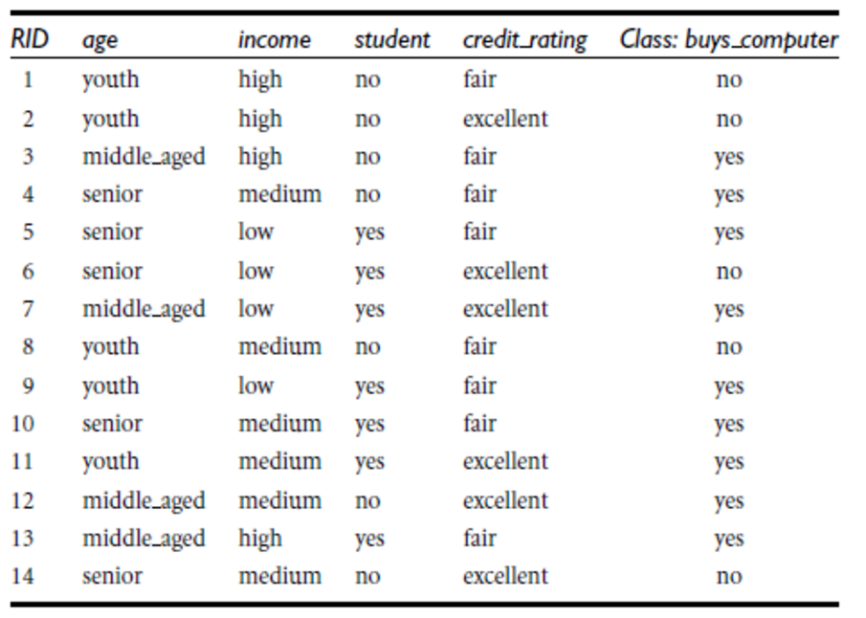
**（解释）**这张图明显是比我们根据规则手画的决策树优化了，根据年龄判断了直接就看是否为学生就可以达到分类目的。无需根据年龄🡪收入🡪学生-🡪是否购买来判断。

**（总结）**根据信息熵和信息增益得到的决策树和之前利用自己制定规则得到的决策树，有明显的不同，因此，利用信息熵和信息增益可以得到分类效果更优的决策树。



# 决策树解决电商数据预测购买及补充问题

## 6.1电商数据集实战



方法1：

age income student credit\_rating Class:buy\_computer

0 1 3 0 1 0

1 1 3 0 0 0

2 2 3 0 1 1

3 3 2 0 1 1

4 3 1 1 1 1

5 3 1 1 0 0

6 2 1 1 0 1

7 1 2 0 1 0

8 1 1 1 1 0

9 3 2 1 1 1

10 1 2 1 0 1

11 2 2 0 0 1

12 2 3 1 1 1

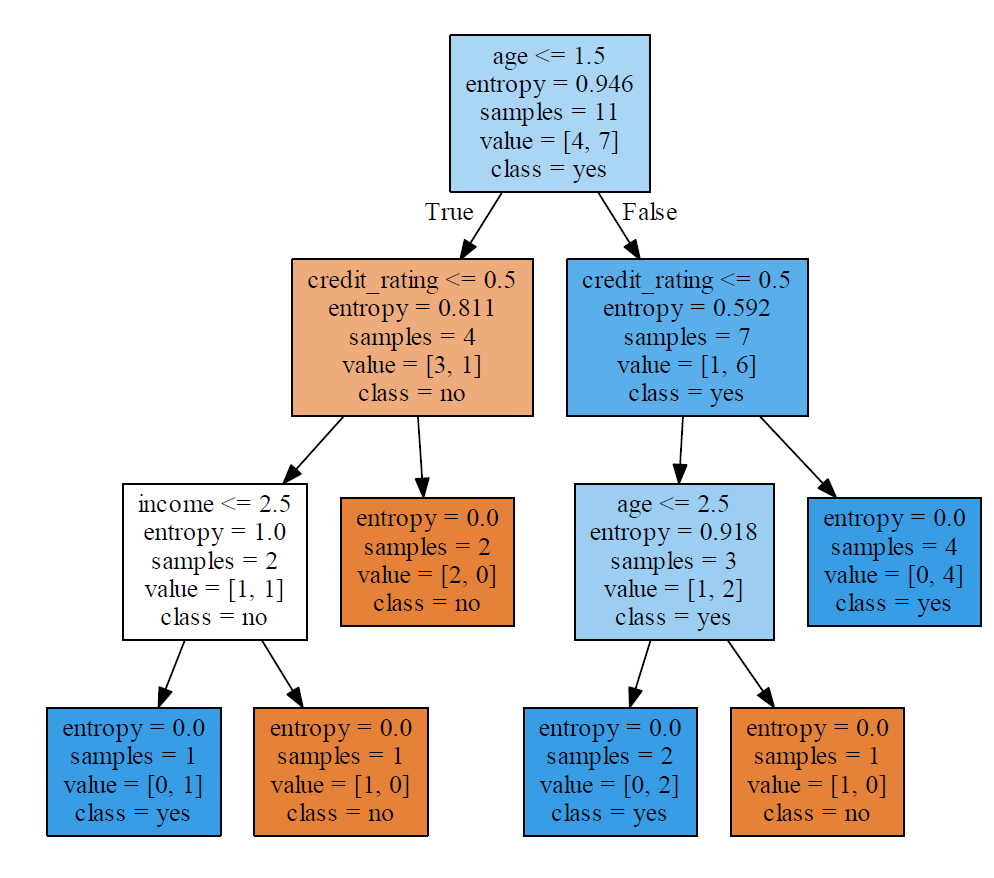
13 3 2 0 0 0

方案2：labelencoder+onehotencoder

### 6.1.1构建机器学习流程

**#1.导入数据集  
import pandas as pd  
buyData = pd.read\_csv("./buy.csv", sep=",")  
#2.数据进行简单的探索性分析  
print("Dataset head:\n",buyData.head(3))  
print("Dataset shape:",buyData.shape)  
print("Dataset ndim:",buyData.ndim)  
print("DateSet index:",buyData.index)  
print("DateSet columns:",buyData.columns)  
print("DataSet info:",buyData.info())  
# age 14 non-null int64  
# income 14 non-null int64  
# student 14 non-null int64  
# credit\_rating 14 non-null int64  
# Class:buy\_computer 14 non-null int64  
# dtypes: int64(5)  
# memory usage: 640.0 bytes  
#3.将数据切分成训练集和测试集  
#如果你是用sklearn版本是0.18之前切分数据api是cross\_validation  
#如果定义的是0.18以后的版本，使用的是model\_selection  
# age income student credit\_rating Class:buy\_computer  
# 0 1 3 0 1 0  
# 1 1 3 0 0 0  
# 2 2 3 0 1 1  
X=buyData.drop("Class:buy\_computer",axis=1)  
y=buyData["Class:buy\_computer"]  
print(X)  
print(y)  
from sklearn.model\_selection import train\_test\_split  
#test\_size将训练集切分成测试集和训练集，其中测试集占比0.2=20%  
#random\_state是随机数种子，能够保证每次结果的可重复性  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,random\_state=22,test\_size=0.2)  
print(X\_train)  
# age income student credit\_rating  
# 1 1 3 0 0  
# 8 1 1 1 1  
# 11 2 2 0 0  
# 10 1 2 1 0  
# 3 3 2 0 1  
# 9 3 2 1 1  
# 6 2 1 1 0  
# 12 2 3 1 1  
# 0 1 3 0 1  
# 4 3 1 1 1  
# 5 3 1 1 0  
print(X\_test)  
# age income student credit\_rating  
# 7 1 2 0 1  
# 13 3 2 0 0  
# 2 2 3 0 1  
#4.使用训练集训练模型  
from sklearn.tree import DecisionTreeClassifier  
dtc=DecisionTreeClassifier(criterion='entropy')  
# DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None,  
# max\_features=None, max\_leaf\_nodes=None,  
# min\_impurity\_decrease=0.0, min\_impurity\_split=None,  
# min\_samples\_leaf=1, min\_samples\_split=2,  
# min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=None,  
# splitter='best')  
print(dtc.fit(X\_train,y\_train))  
#5.使用模型进行预测  
y\_pred=dtc.predict(X\_test)  
#6.使用测试集测试模型情况  
print("model in trainset score is:%.2f%%"%(dtc.score(X\_train,y\_train)))  
print("model in testset score is:%.2f%%"%(dtc.score(X\_test,y\_test)))  
#混淆矩阵  
from sklearn.metrics import confusion\_matrix  
print(confusion\_matrix(y\_true=y\_test,y\_pred=y\_pred))  
#7.保存模型  
from sklearn.externals import joblib  
joblib.dump(dtc,"buyDataSet.pkl")**

### 6.1.2结果可视化展示



简记：泰坦尼克号：1500多人，如今通过计算机模拟和分析找出潜在数据背后的生还逻辑。

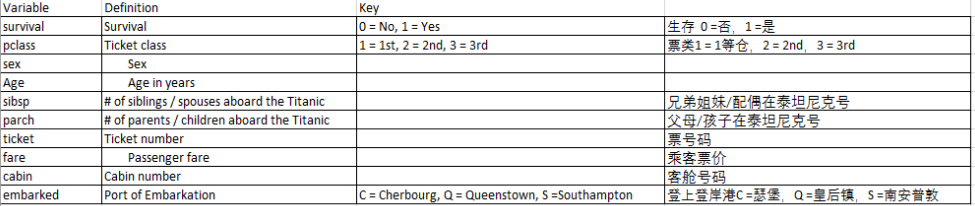
大家都看过电影泰坦尼克号，是历史上一件家喻户晓的灾难性事件；泰坦尼克号沉船事故，1912年，当时属于英国的世界级豪华客轮泰坦尼克号，因在处女航行中不幸撞上北大西洋冰山而沉没。这场事故使得1500多名乘客患难。后来，这场震惊世界的惨剧被详细地调查，而且遇难乘客的信息也逐渐被披露。在当时救援条件下，无法短时间内确认每位乘客生还的可能性。而今，许多科学家视图通过计算机模拟和分析找出潜藏在数据背后的生还逻辑。我们也尝试通过代码，揭开尘封了100多年的数据的面纱。

数据示例：

<http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt>



字段信息的备注：



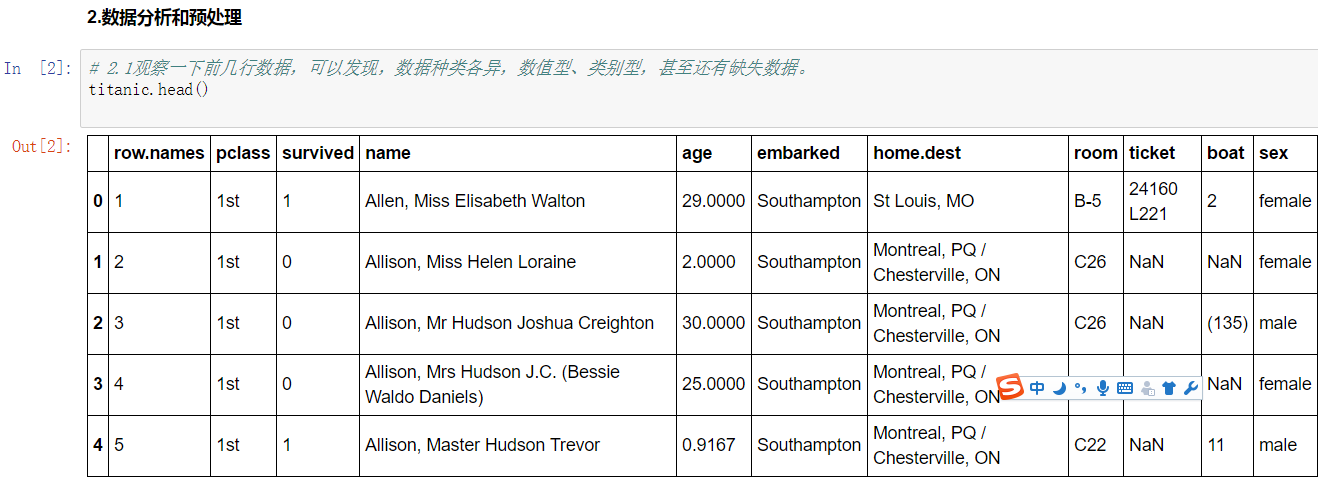
机器学习任务：预测泰坦尼克号乘客能否存活 预测模型 分类问题

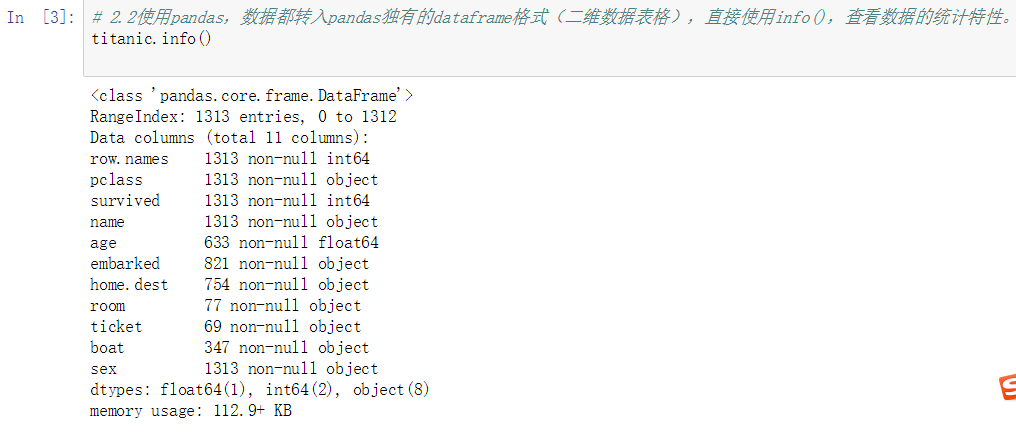
## 6.2泰坦尼克号沉船问题

### 6.1数据导入



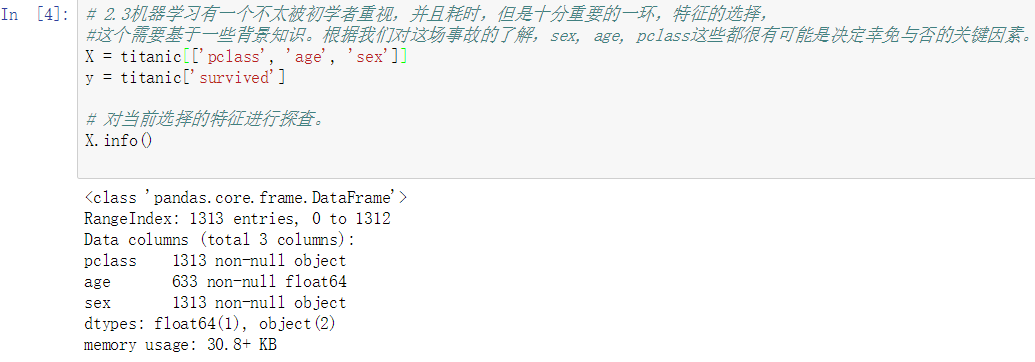
### 6.2数据分析和预处理

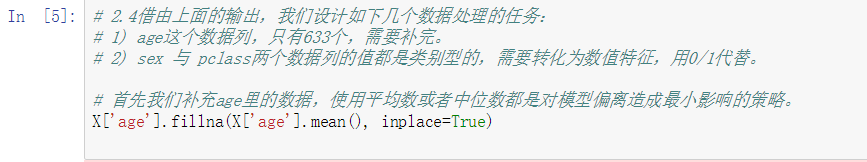


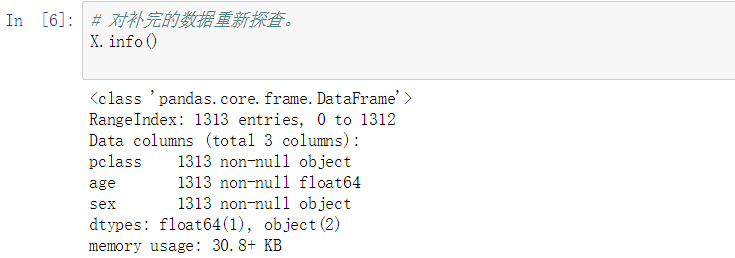


#### 注：该数据共有1313条乘客信息，并且有些特征数据是完整的，比如name和pclass，有些是缺失的；有些是数值型的，有些是字符串型的

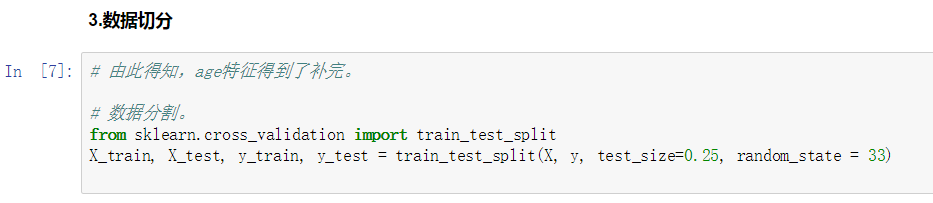
#### 由于数据比较久远，难免会有信息的丢失和不完整，也有一些数据没有量化。因此，在使用决策树模型之前，需要对数据做一些预处理和分析工作。



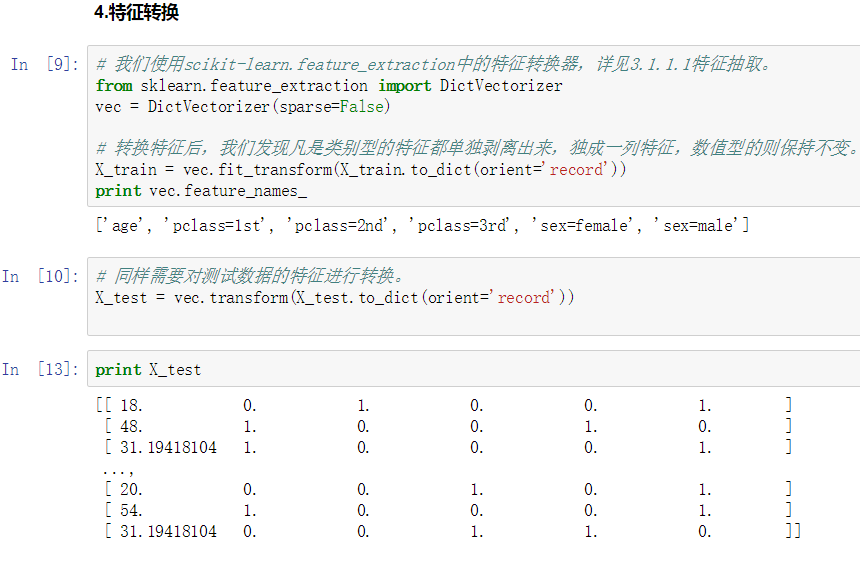




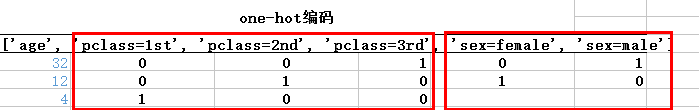
### 6.3数据切分



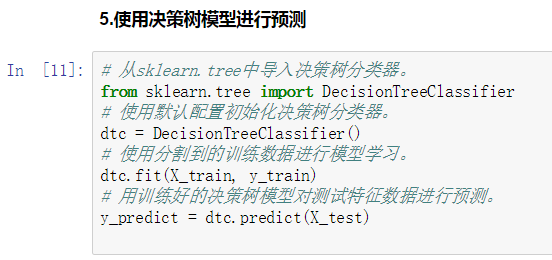
### 6.4数据特征转换



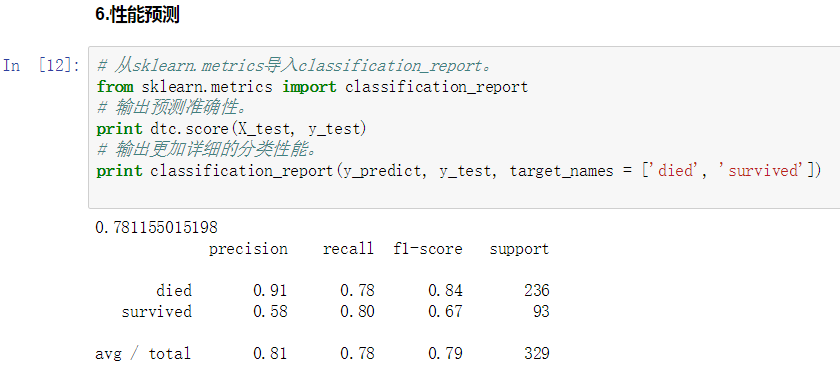
注：one-hot编码



### 6.5使用决策树模型进行预测



### 6.6决策树模型对泰坦尼克号乘客是否生还的预测性能



总结:决策树模型总体在测试集上的预测性能约为78.12%，大家观察能看到模型对于预测遇难者方面的性能达到了91%，但是识别生还者的精确率只有58%，这也是需要提高的地方。

### 6.7决策树可视化

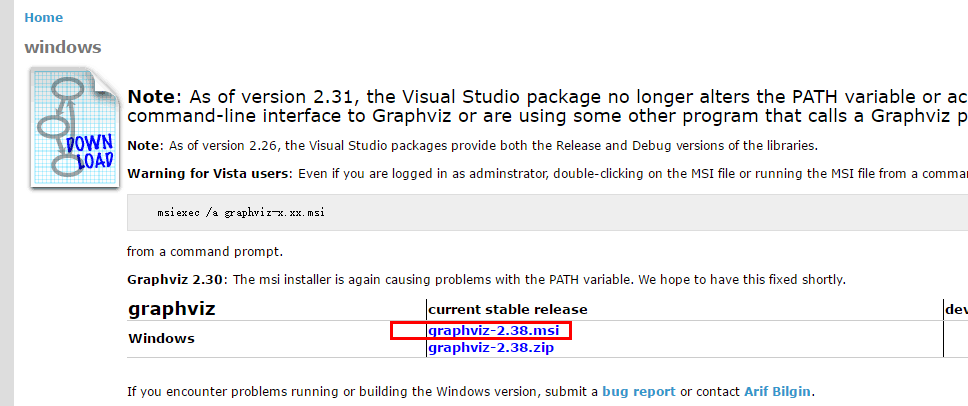
Graphviz是贝尔实验室开发的一个开源的工具包，用于绘制结构化的图形网络，支持多种输出格式，如常见的图片格式，SVG，PDF等，且支持Linux和windows平台。

通过Graphviz 的地址下载该软件：<http://www.graphviz.org/>

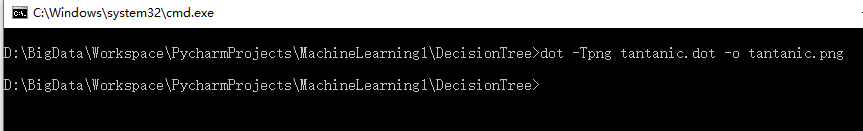
  配置本地环境变量，通过命令dot -Tpng iris.dot -o output.png将dot文件转化至png图片可视化决策树。

***安装的过程：***

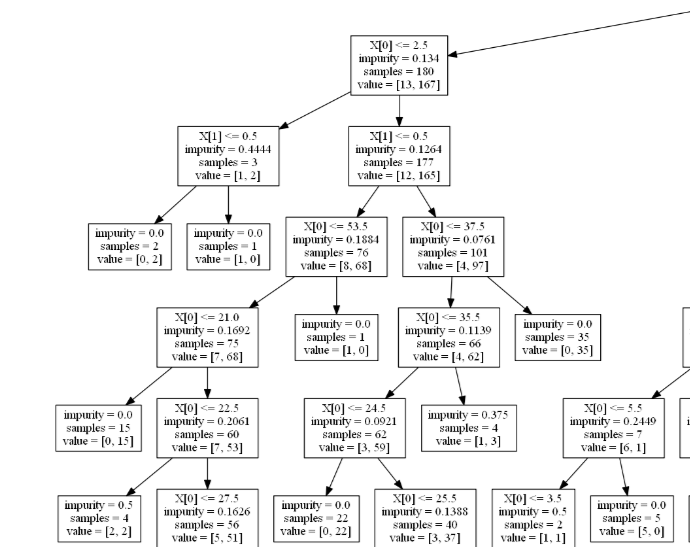
1. <http://www.graphviz.org/Download_windows.php>安装

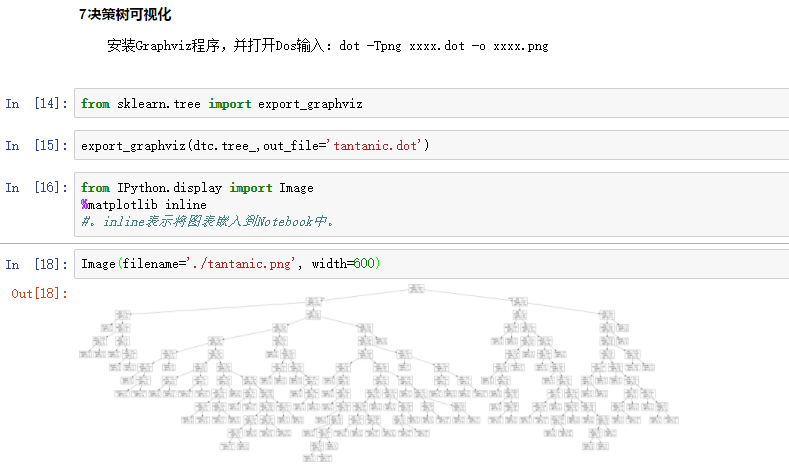


（2）配置环境变量并在Dos中输入命令即可完成将dot文件转化为png文件。



（3）查看图片内容





总结：

相比其他学习模型，决策树在模型描述上有巨大的优势，决策树的逻辑推断非常直观，具有清晰的可解释性，也有很方便的模型的可视化。在决策树的使用中，无需考虑对数据量化和标准化，就能达到比较好的识别率。

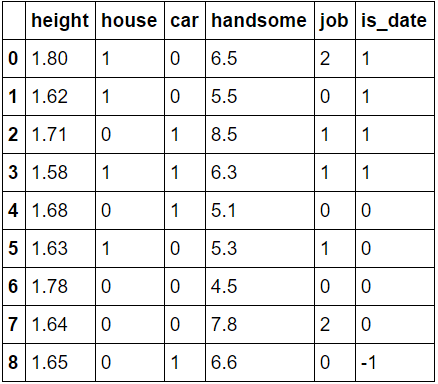
# 7.决策树算法的两个经典案例

在刚讲决策树之前，我们引入了相亲的例子，接下来我们就利用模拟的一些数据，并将数据的某些字段转化为算法或程序能处理的数值，利用决策树来进行分类。

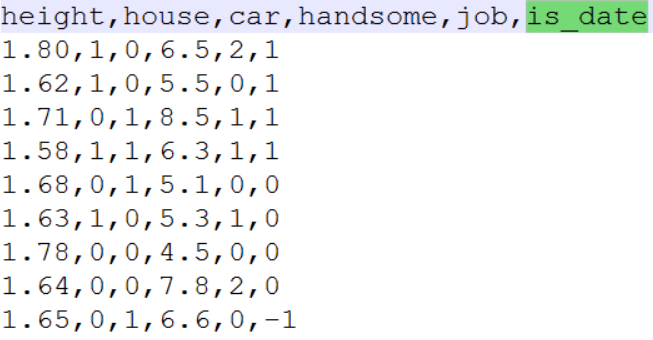
表5-1 相亲数据

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序号 | 身高 | 房子 | 车子 | 长相 | 工作 | 约否 |
| 1 | 1.8 | 有 | 无 | 6.5 | Fact | 约 |
| 2 | 1.62 | 有 | 无 | 5.5 | IT | 约 |
| 3 | 1.71 | 无 | 无 | 8.5 | Bank | 约 |
| 4 | 1.58 | 有 | 有 | 6.3 | Bank | 约 |
| 5 | 1.68 | 无 | 有 | 5.1 | IT | 不约 |
| 6 | 1.63 | 有 | 无 | 5.3 | Bank | 不约 |
| 7 | 1.78 | 无 | 无 | 4.5 | It | 不约 |
| 8 | 1.64 | 无 | 无 | 7.8 | Fact | 不约 |
| 9 | 1.65 | 有 | 有 | 6.6 | bank | 约吗？ |

转化为可处理的数据：

9

数据集如下：



**接下来通过scikit-learn实现上述分类需求。**

## 7.1.Scikit-Learn库实现**相亲约会**的例子

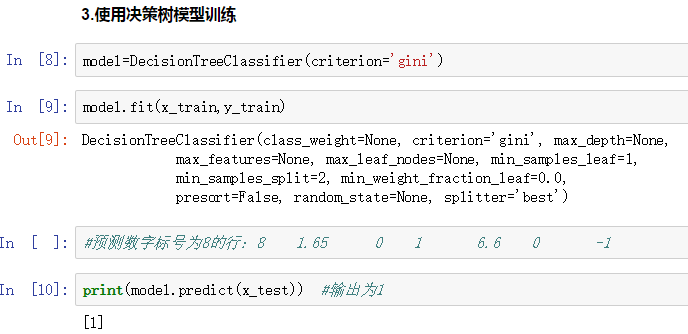
### 1.导包并加载数据



### 2.划分数据集为训练集和测试集



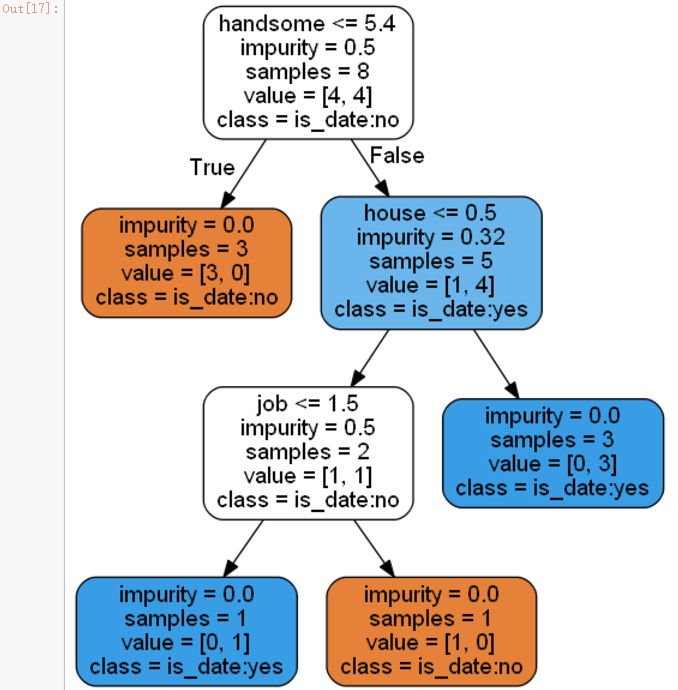
### 3.使用决策树进行模型训练



### 4.决策树模型可视化



可视化结果如下：



### 5.对上述示例的决策树过程进行分析

#### 5.1通过scikit-learn中提供的export\_graphviz将决策树可视化

在实验环境中添加如下代码：

from sklearn.tree import export\_graphviz

export\_graphviz(model.tree\_,out\_file='tree.dot',

feature\_names=df.columns,

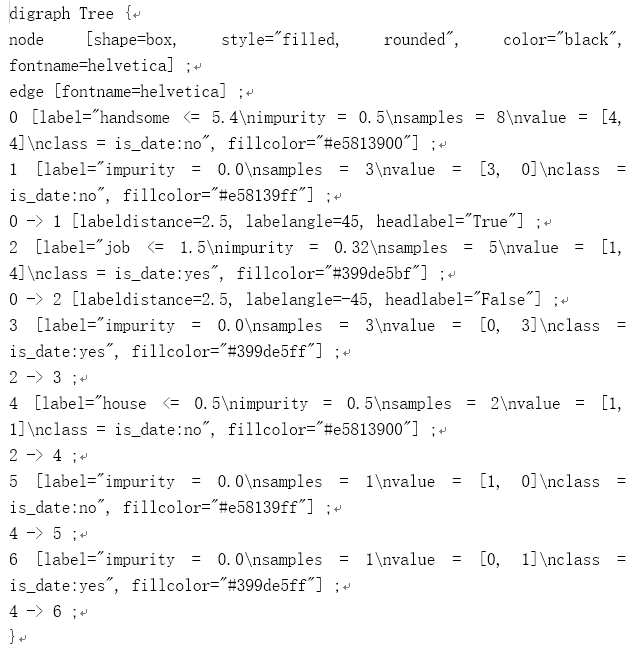
max\_depth=None,

class\_names=["is\_date:no","is\_date:yes"],

rounded=True,

filled=True,)

#### 5.2 执行上述代码将会生成tree.dot文件，里面包含了建立树所需要的信息



#### 5.3 在本地安装 Graphviz

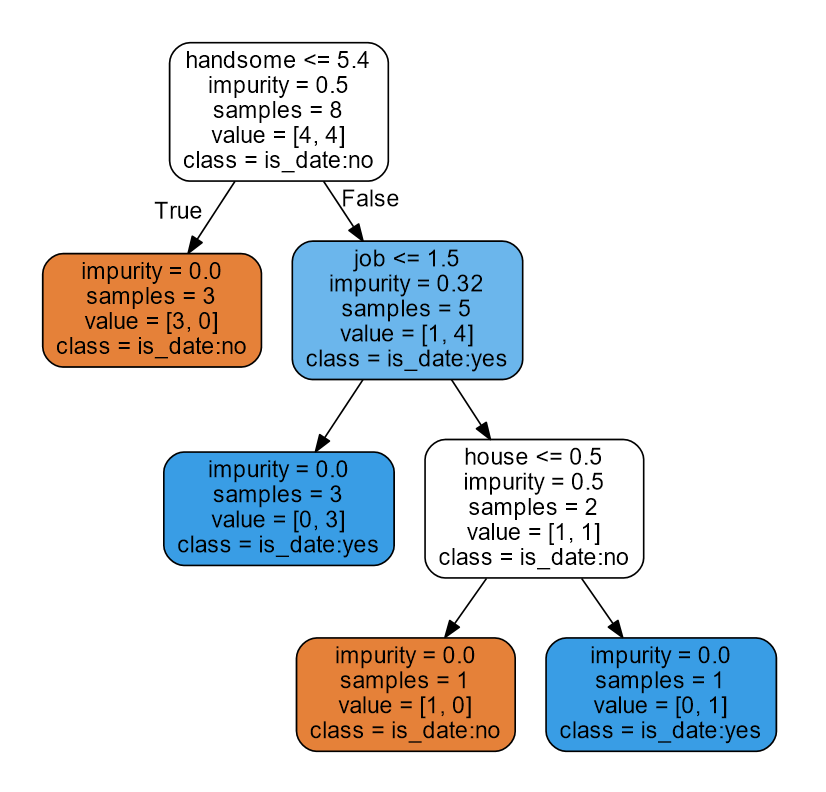
通过Graphviz 的地址下载该软件：<http://www.graphviz.org/>

  配置本地环境变量，通过命令dot -Tpdf iris.dot -o outpu.pdf将dot文件转化至pdf可视化决策树。



在本地会产生一个pdf文件，包含了回执的图示，如下图

#### 5.4 分析图



**class\_names : list of strings, bool or None, optional (default=None)  
 Names of each of the target classes in ascending numerical order.  
 Only relevant for classification and not supported for multi-output.  
 If ``True``, shows a symbolic representation of the class name.**

class\_names=["is\_date:no","is\_date:yes"],这里为什么指定no代表0，yes代表1呢？这是因为在class\_names中默认以数值递增的顺序进行对应，也是就是0对应is\_date:no.

注：对上述图像解析几点如下：

第一行为决策条件（非叶子节点），比如根节点handsome《5.4条件为真走左边，为假走右边。

* Impurity表示当前数据集的基尼不纯度
* Sample表示当前数据集的样本数
* Value表示当前数据集中各个类别（按类别顺序从小到大排序，第0类，第1类…）的数量，如果value中的值相等，当前节点没有作色，也就是白色。
* Class表示当前数据中的多数类别，如果value中各个值相等，则按顺序取值

#### 5.5分析图中的逻辑

长相小于或等于5.4的，一定不约

不满足前面条件，但有房子的，一定约

不满足前面条件，但有车，约，没有车的，不约

2.6对于待测数据

长相大于5.4，不满则规则1，则继续下一条。

没有房子，不满足第二条规则2，继续吓一条

有车，符合第三条规则，最后的结果为约，也就是上述的结果1

### 改进例子

import pandas as pd  
#加载数据  
love\_file=pd.read\_csv("./SklearnTest.txt")  
#数据集(切分为训练集和测试集)和新数据  
data,newData=love\_file.query("is\_date!=-1"),love\_file.query("is\_date==-1")  
#将数据集切分成X特征和Y类别标签  
X\_data=data.drop(["is\_date"],axis=1)  
y\_data=data["is\_date"]  
#切分数据集  
from sklearn.cross\_validation import train\_test\_split  
#random——state随机数种子-保证每一次切分数据集结果可重复性  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_data, y\_data, test\_size=0.33, random\_state=1)  
#建立决策树模型  
from sklearn.tree import DecisionTreeClassifier  
dtc=DecisionTreeClassifier(criterion="entropy")  
#训练模型使用的是fit方法---训练模型  
print(dtc.fit(X\_train,y\_train))  
#预测新数据的结果  
y\_pred=dtc.predict(X\_test)  
#模型校验  
print("model in train set score:",dtc.score(X\_train,y\_train))  
print("model in test set score:",dtc.score(X\_test,y\_test))  
# model in train set score: 1.0  
# model in test set score: 0.3333333333333333  
from sklearn.metrics import confusion\_matrix  
print(confusion\_matrix(y\_test,y\_pred))  
#新数据  
X\_new\_data=newData.drop(["is\_date"],axis=1) #y\_new\_data  
print(dtc.predict(X\_new\_data))  
#可视化处理  
from sklearn.tree import export\_graphviz  
export\_graphviz(dtc.tree\_,out\_file="love.dot",filled=True,feature\_names=love\_file.columns,class\_names=["no","yes"])

### 改进例子

import pandas as pd  
#加载数据  
love\_file=pd.read\_csv("./SklearnTest.txt")  
#1.数据集(切分为训练集和测试集)和新数据  
data,newData=love\_file.query("is\_date!=-1"),love\_file.query("is\_date==-1")  
#将数据集切分成X特征和Y类别标签  
X\_data=data.drop(["is\_date"],axis=1)  
y\_data=data["is\_date"]  
#2.切分数据集  
from sklearn.cross\_validation import train\_test\_split  
#random——state随机数种子-保证每一次切分数据集结果可重复性  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_data, y\_data, test\_size=0.33, random\_state=1)  
#3.数据处理  
from sklearn.preprocessing import StandardScaler  
sc=StandardScaler()  
#对训练数据集先进行fit操作在进行transfrom操作  
#fit操作相当于收集所有参数信息，transfrom将所有的数据减去均值除以方差  
X\_train\_std=sc.fit\_transform(X\_train)  
#对于测试数据集来讲只需要使用transform方法进行归一化  
X\_test\_std=sc.transform(X\_test)  
#4.建立决策树模型  
from sklearn.tree import DecisionTreeClassifier  
dtc=DecisionTreeClassifier(criterion="entropy")  
#训练模型使用的是fit方法---训练模型  
print(dtc.fit(X\_train\_std,y\_train))  
#5.预测新数据的结果  
y\_pred=dtc.predict(X\_test)  
#6.模型校验  
print("model in train set score:",dtc.score(X\_train\_std,y\_train))  
print("model in test set score:",dtc.score(X\_test\_std,y\_test))

## 7.2Iris数据集的例子

数据集简介：iris以鸢尾花的特征作为数据来源，数据集包含150个样本，分为3类，每类50个数据，每个数据包含4个属性，是在数据挖掘、数据分类中非常常用的测试集、训练集。

三类分别为:setosa, versicolor, virginica

数据包含4个独立的属性,这些属性变量测量植物的花朵,比如萼片和花瓣的长度等。

机器学习的目的：预测iris数据集类别，预测模型，分类问题

**（选讲）**DecisionTreeClassifier是能够在数据集上执行多类分类的类。

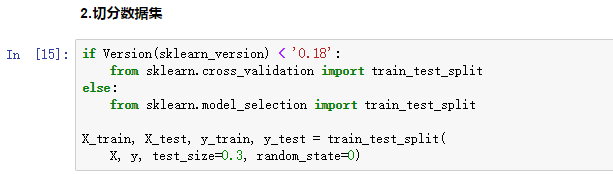
与其他分类器一样，DecisionTreeClassifier将输入两个数组:数组X，稀疏或密集，大小为[n\_samples，n\_features]，保存训练样本，以及整数值的数组Y，大小[n\_samples]，持有类标签训练样本。

### 导包和加载数据集

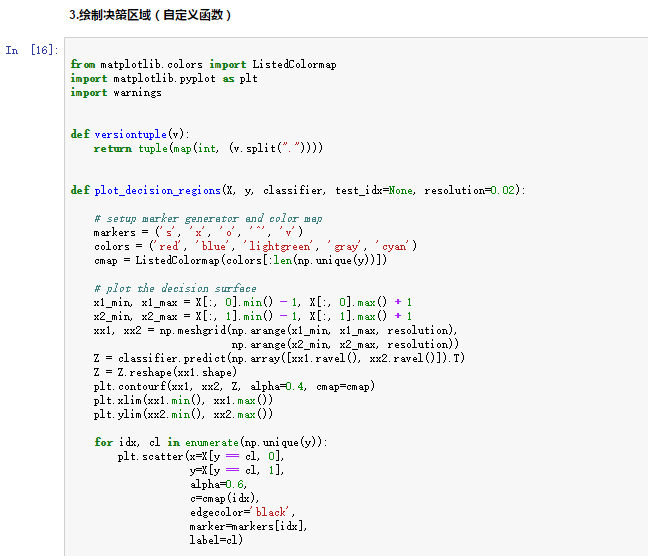
从scikit学习加载Iris数据集。 这里，第三列表示花瓣长度，第四列表示花样的花瓣宽度。 这些类已经转换为整数标签，其中0 = Iris-Setosa，1 = Iris-Versicolor，2 = Iris-Virginica。

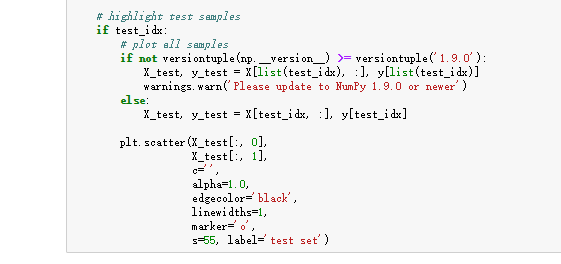


### 切分数据集



### 绘制决策区域函数

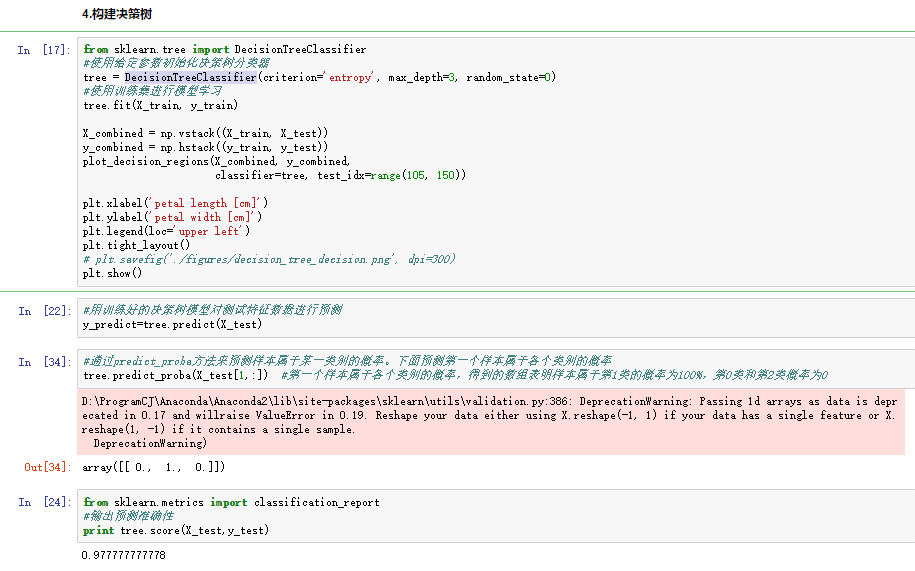




### 构建决策树

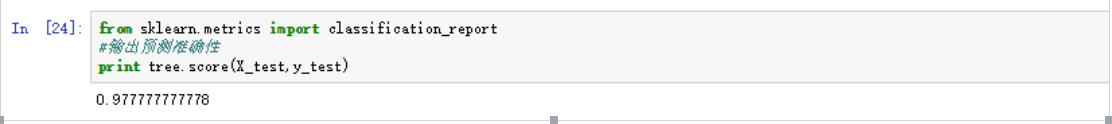
注意：hstack()    Stack arrays in sequence horizontally (column wise).

vsplit ()   Split array into a list of multiple sub-arrays vertically.

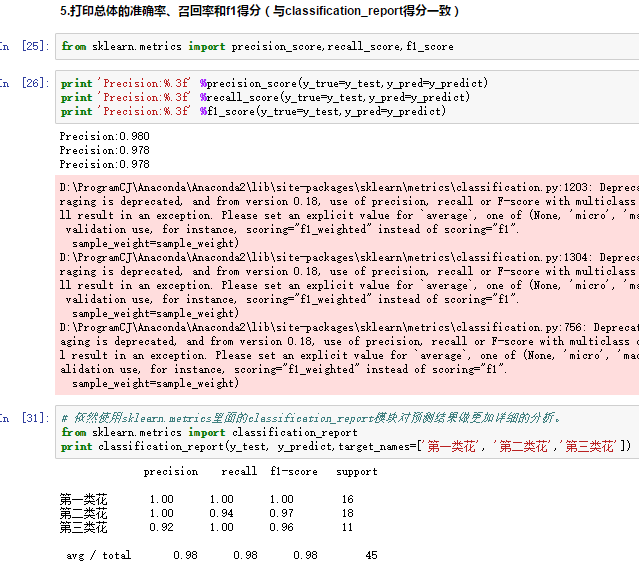


### 决策树性能测试

预测准确率：



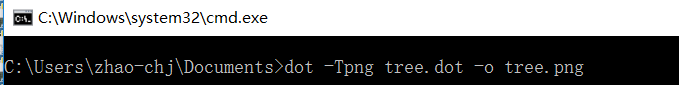
下面分别调用准确率、召回率和f1值的函数查看具体值，也可以直接调用classification\_report函数，直接生成具体的测试值。代码如下：



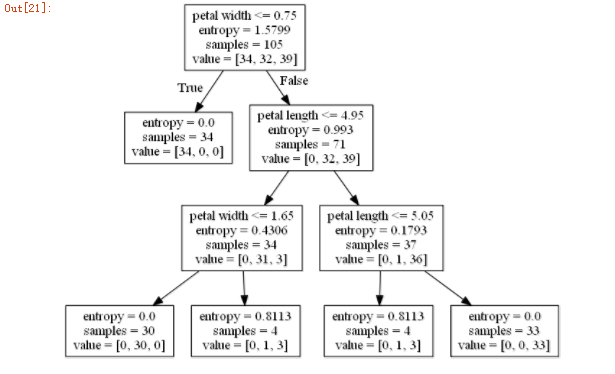
### 决策树可视化



在Dos中执行命令生成PNG的图片格式。



图片显示如下：



通过观察GraphViz创建图像，可以很好地回溯决策树在训练集上对各个节点进行划分的过程。最初，决策树根节点包含105个样本，以花瓣宽度《0.75厘米为界限，将其划分为分别包含34和71个样本的两个子节点。 第一次划分后，可以看到：左子树的样本均来自Iris-Setosa类（熵=0），已经无需再划分。在右子树上进一步划分，直到将Iris-Versicolor和Iris-Virginica两类分开。

# 8.分类决策树SKlearn的API实现

## 7.1API介绍

介绍时候可以使用谷歌浏览器，将翻译的部分让同学们有一个大概的了解。

决策树文档：<http://scikit-learn.org/stable/modules/tree.html#classification>

决策树分类器的简介：

<http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.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)[source]¶

接下来分别分析参数

## 7.2API的参数详解

1. criterion：一个字符串，指定切分质量的评价准则。可以为如下两个参数：

Gini：表示切分时评价准则时Gini系数。

Entropy：表示切分的时候评价准则是信息增益

1. splitter：一个字符串，指定切分原则，可以为如下：

best：表示选择最优的切分

random：表示随机切分

1. max\_features:可以为整数、浮点、字符串或者None，指定了寻找best split时考虑的特征数量。

如果是整数，则每次切分只考虑max\_features个特征。

如果是浮点数，则每次切分只考虑max\_features\*n\_features个特征。

如果是字符串auto或者sqrt，则max\_features等于sqrt(n\_feautures)

如果是字符串log2，则max\_features等于log2（n\_features）

如果是none，则max\_features等于n\_features.

1. max\_depth：可以为整数或者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的，即叶子结点中所有的样本点都属于一个类，或者叶子中包含小于min\_samples\_split个样本点。

1. min\_samples\_split:为整数，指定每个内部节点（非叶子节点）包含的最少样本数。
2. min\_samples\_leaf为整数，指定每个叶子节点包含的最少的样本数。
3. min\_weight\_fraction\_leaf:为浮点数，叶子结点中样本的最小权重系数。
4. min\_leaf\_nodes:为整数或None，指定最大的叶节点数量。

如果为None，此时叶子节点数量不限制。

如果为非None，则max\_depth被忽略。

1. class\_weight:为一个字典、字典的列表、字符串‘balanced’，或者None，它指定了分类的权重。权重的形式为：{class\_label:weight}

如果是None，则每个分类的权重都为1.

字符串balanced表示分类权重是样本中各分类出现频率的反比。

1. random\_state:一个整数或者一个Randomstate实例，或者None。

如果为整数，则它指定了随机数生成器的种子。

如果为Randomstate实例，则指定了随机数生成器。

如果为None，则使用默认的随机数生成器。

1. presort：一个布尔值，指定是否提前排序数据，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。

## 7.3API的属性详解

共7各属性：

(1)classes\_:分类的标签值

(2)feature\_importances\_:给出了特征的重要程度。该值越高，则特征越重要（也称之为Gini\_importance）

(3)max\_feature\_:max\_feature的推断值。

(4)n\_classes\_：给出了分类的数量

(5)n\_feautures\_:执行fit之后特征的数量

(6)n\_outputs\_:执行fit之后输出的数量

(7)tree\_(tree):一个Tree对象，即底层的决策树。:

## 7.4API的方法详解

共有五种方法：

1. fit(X,y):训练模型
2. predict(X,)用模型进行预测，返回预测值
3. predict\_log\_proba(X):返回一个数组，数组的元素依次是X预测为各个类别的概率的对数值。
4. predict\_proba(X)：返回一个数组，数组的元素依次时X预测为各个类别的概率值。
5. score(X,y):返回在（X，y）上预测的准确率（accurary）的平均值。

# 9.Python原生实现代码：

## 9.1ID3算法的Python实现

### 1.ID3算法实现

ID3DTree.py

**>>># -\*- coding: utf-8 -\*-**

**from numpy import \***

**import math**

**import copy**

**import cPickle as pickle**

**class ID3DTree(object):**

**def \_\_init\_\_(self):**

**self.tree={}**

**self.dataSet=[]**

**self.labels=[]**

**def loadDataSet(self,path,labels):**

**recordlist = []**

**fp = open(path,"rb") # 读取文件内容**

**content = fp.read()**

**fp.close()**

**rowlist = content.splitlines() # 按行转换为一维表**

**recordlist=[row.split("\t") for row in rowlist if row.strip()]**

**self.dataSet = recordlist**

**self.labels = labels**

**def train(self):**

**labels = copy.deepcopy(self.labels)**

**self.tree = self.buildTree(self.dataSet,labels)**

**# 创建决策树主程序**

**def buildTree(self,dataSet,labels):**

**cateList = [data[-1] for data in dataSet] # 抽取源数据集的决策标签列**

**# 程序终止条件1: 如果classList只有一种决策标签，停止划分，返回这个决策标签**

**if cateList.count(cateList[0]) == len(cateList):**

**return cateList[0]**

**# 程序终止条件2: 如果数据集的第一个决策标签只有一个 返回这个决策标签**

**if len(dataSet[0]) == 1:**

**return self.maxCate(cateList)**

**# 算法核心：**

**bestFeat = self.getBestFeat(dataSet) # 返回数据集的最优特征轴：**

**bestFeatLabel = labels[bestFeat]**

**tree = {bestFeatLabel:{}}**

**del(labels[bestFeat])**

**# 抽取最优特征轴的列向量**

**uniqueVals = set([data[bestFeat] for data in dataSet]) # 去重**

**for value in uniqueVals:**

**subLabels = labels[:] #将删除后的特征类别集建立子类别集**

**splitDataset = self.splitDataSet(dataSet, bestFeat, value) # 按最优特征列和值分割数据集**

**subTree = self.buildTree(splitDataset,subLabels) # 构建子树**

**tree[bestFeatLabel][value] = subTree**

**return tree**

**#计算出现最多的列别标签**

**def maxCate(self,catelist): # 计算出现最多的类别标签**

**items = dict([(catelist.count(i), i) for i in catelist])**

**return items[max(items.keys())]**

**#计算最优特征**

**def getBestFeat(self,dataSet):**

**# 计算特征向量维，其中最后一列用于类别标签，因此要减去**

**numFeatures = len(dataSet[0]) - 1 # 特征向量维数=行向量维度-1**

**baseEntropy = self.computeEntropy(dataSet) # 基础熵：源数据的香农熵**

**bestInfoGain = 0.0; # 初始化最优的信息增益**

**bestFeature = -1 # 初始化最优的特征轴**

**# 外循环：遍历数据集各列,计算最优特征轴**

**# i 为数据集列索引：取值范围 0~(numFeatures-1)**

**for i in xrange(numFeatures): # 抽取第i列的列向量 uniqueVals = set([data[i] for data in dataSet]) # 去重：该列的唯一值集 newEntropy = 0.0 # 初始化该列的香农熵**

**for value in uniqueVals: # 内循环：按列和唯一值计算香农熵 subDataSet = self.splitDataSet(dataSet, i, value) # 按选定列i和唯一值分隔数据集**

**prob = len(subDataSet)/float(len(dataSet))**

**newEntropy += prob \* self.computeEntropy(subDataSet) infoGain = baseEntropy - newEntropy # 计算最大增益**

**if (infoGain > bestInfoGain): # 如果信息增益>0; bestInfoGain = infoGain**

**# 用当前信息增益值替代之前的最优增益值 bestFeature = i # 重置最优特征为当前列**

**return bestFeature**

**#计算信息熵**

**def computeEntropy(self,dataSet): # 计算香农熵**

**datalen = float(len(dataSet))**

**cateList = [data[-1] for data in dataSet] # 从数据集中得到类别标签**

**items = dict([(i,cateList.count(i)) for i in cateList])**

**# 得到类别为key，出现次数value的字典**

**infoEntropy = 0.0 # 初始化香农熵 for key in items: # 计算香农熵**

**prob = float(items[key])/datalen**

**infoEntropy -= prob \* math.log(prob,2) # 香农熵：= - p\*log2(p) --infoEntropy = -prob \* log(prob,2)**

**return infoEntropy**

**# 分隔数据集：删除特征轴所在的数据列，返回剩余的数据集**

**# dataSet：数据集; axis：特征轴; value：特征轴的取值**

**def splitDataSet(self, dataSet, axis, value):**

**rtnList = []**

**for featVec in dataSet:**

**if featVec[axis] == value:**

**rFeatVec = featVec[:axis]**

**# list操作 提取0~(axis-1)的元素**

**rFeatVec.extend(featVec[axis+1:])**

**# list操作 将特征轴（列）之后的元素加回**

**rtnList.append(rFeatVec)**

**return rtnList**

**#决策树分类器代码**

**def predict(self,inputTree,featLabels,testVec): # 分类器**

**root = inputTree.keys()[0] # 树根节点**

**secondDict = inputTree[root] # value-子树结构或分类标签**

**featIndex = featLabels.index(root) # 根节点在分类标签集中的位置**

**key = testVec[featIndex] # 测试集数组取值**

**valueOfFeat = secondDict[key] #**

**if isinstance(valueOfFeat, dict):**

**classLabel = self.predict(valueOfFeat, featLabels, testVec) # 递归分类**

**else: classLabel = valueOfFeat**

**return classLabel**

**# 存储树到文件**

**def storeTree(self,inputTree,filename):**

**fw = open(filename,'w')**

**pickle.dump(inputTree,fw)**

**fw.close()**

**# 从文件抓取树**

**def grabTree(self,filename):**

**fr = open(filename)**

**return pickle.load(fr)**

### 2.构建数据集

builddataset.py

**# -\*- coding: utf-8 -\*-**

**import sys**

**import os**

**from numpy import \***

**# 设置utf-8 unicode环境**

**reload(sys)**

**sys.setdefaultencoding('utf-8')**

**#构建输入的数据集**

**labels=["年龄","收入","学生","信誉"]**

**dataset = [[0,0,0,0,"no"],[0,0,0,1,"no"],[0,1,0,0,"no"],[0,2,1,0,"yes"],[0,1,1,1,"yes"],**

**[1,0,0,0,"yes"],[1,2,1,1,"yes"], [1,1,0,1,"yes"],[1,0,1,0,"yes"],[2,1,0,0,"yes"],[2,2,1,0,"yes"],**

**[2,2,1,1,"no"],[2,1,1,0,"yes"],[2,1,0,1,"no"]]**

**numlist = [64 ,64 ,128,64 ,64 ,128,64 ,32 ,32 ,60 ,64 ,64 ,132,64 ]**

**print mat(dataset).T**

**datalines =[]**

**for element,num in zip(dataset,numlist):**

**liststr =""**

**for cell in element:**

**liststr += str(cell)+"\t"**

**liststr = liststr[:-1]**

**for i in xrange(num):**

**datalines.append(liststr)**

**fp = open("dataset.dat","w")**

**fp.write("\n".join(datalines))**

### 3.训练的决策树的结构信息打印及字典转化为树状的显示

**>>>** # -\*- coding: utf-8 -\*-

from numpy import \*

from math import log

from ID3DTree import \*

import copy

import treePlotter2

import matplotlib.pyplot as plt

dtree = ID3DTree()

dtree.loadDataSet("dataset.dat",["age","revenue","student","credit"])

# dtree.loadDataSet("lenses.txt",['age','prescript','astigmatic','tearRate'])

dtree.train()

print dtree.tree

treePlotter2.createPlot(dtree.tree)

### 4.持久化决策树

# -\*- coding: utf-8 -\*-

# Filename : 07store\_grab.py

from numpy import \*

from math import log

from ID3DTree import \*

import copy

import treePlotter2

import matplotlib.pyplot as plt

dtree = ID3DTree()

dtree.loadDataSet("dataset.dat",["age","revenue","student","credit"])

dtree.train()

dtree.storeTree(dtree.tree,"data.tree")

mytree = dtree.grabTree("data.tree")

print mytree

### 5.测试ID3算法

-\*- coding: utf-8 -\*-

from numpy import \*

from math import log

from ID3DTree import \*

import copy

import treePlotter2

import matplotlib.pyplot as plt

dtree = ID3DTree()

labels = ["age","revenue","student","credit"]

vector = ['0','1','0','0'] # ['0','1','0','0','no']

mytree = dtree.grabTree("data.tree")

print "真实输出 ","no","->","决策树输出",dtree.predict(mytree,labels,vector)

# 补充：基尼指数、误分类率及代码实战

（1）Gini指数（基尼指数）

选择最佳划分的度量通常是根据划分后子女节点的不纯性的程度决定的。一般最佳划分对应于产生最小Gini指标值得点。

公式(1-1)：

P(i|t)表示给定节点t中属于类i的记录所占的比例。

（2）错误率

以2分类问题为例计算信息熵、基尼不纯度和错误率

|  |  |
| --- | --- |
| 节点 | 计数 |
| 类=0 | 1 |
| 类=1 | 5 |

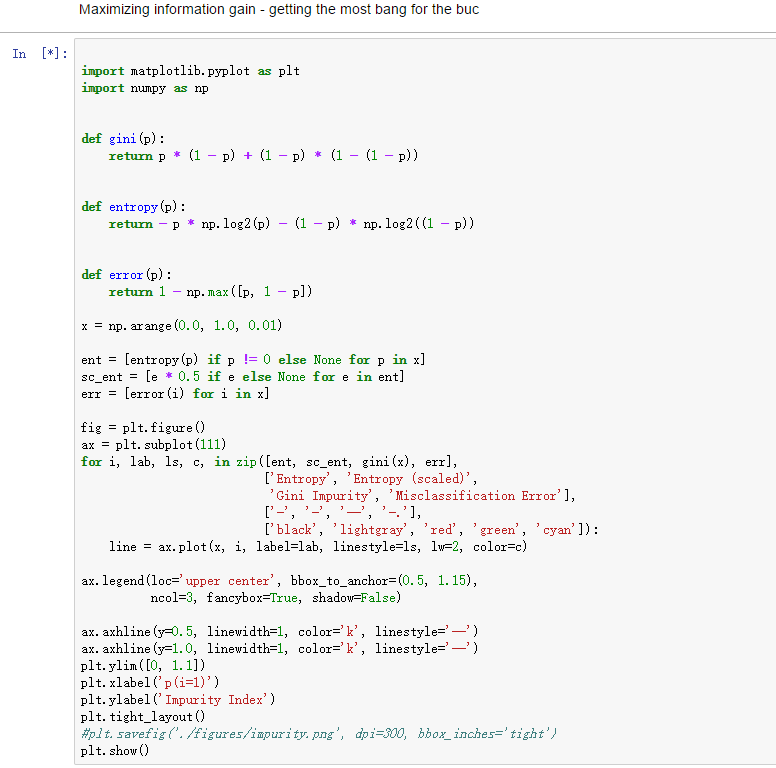
Gini=1-=0.278

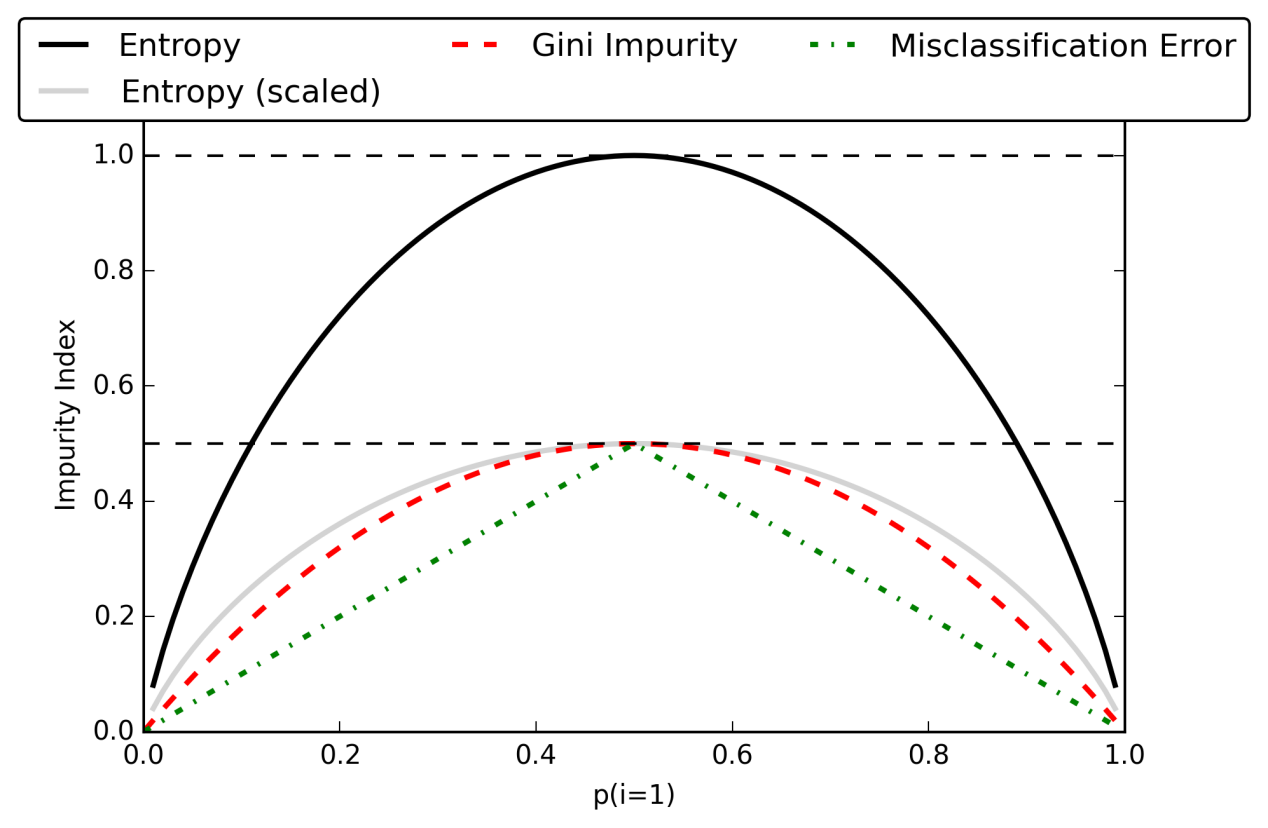
Entropy=-(1/6)log(1/6)-(5/6)log(5/6)=0.650

Error=1-max[1/6,5/6]=0.167

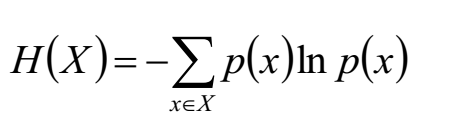
（3）信息增益公式解读

是给定节点的不纯性度量，N是父节点上的记录总数，k是属性值的个数是与子女节点v相关联的记录个数，决策树归纳算法通常选择最大化增益 的测试条件，因为对所有的测试条件来说， 是一个不变的值，所以最大化增益等价于最小化子女节点的不纯性度量的的加权平均值。当选择熵作为不纯性度量时，熵的差就是所谓的信息增益。



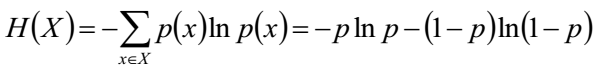


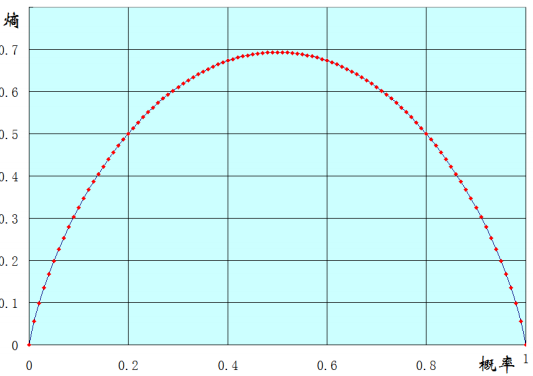
# 补充：最大熵及交叉熵



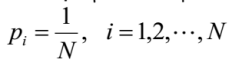
* 底数一般为2或e

## 两点分布的熵

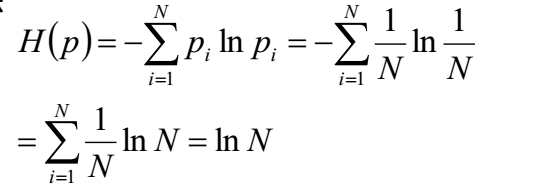




## 均匀分布的信息熵

定义概率：

计算熵：

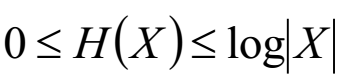


## 最大熵

熵是随机变量不确定性度量，不确定越大，熵越大

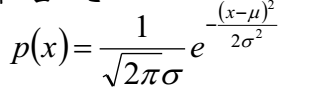
若随机变量退化成定值，熵最小，为0

若随机分布为均匀分布，熵最大

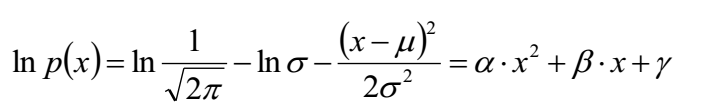


接下来考虑，给定期望和方差的前体下，最大熵模型的形式？

正态分布的概率密度函数：



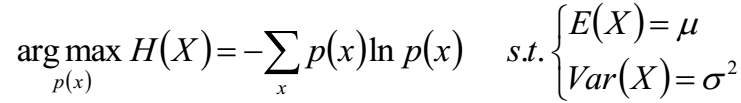
对数正态分布：



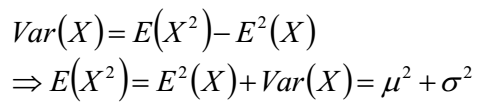
该分布的对数是关于随机变量x的二次函数，根据计算过程的可逆性，如若对数分布能够写成随机变量的二次形式，则该分布必然是正态分布。

给定方差的最大熵分布

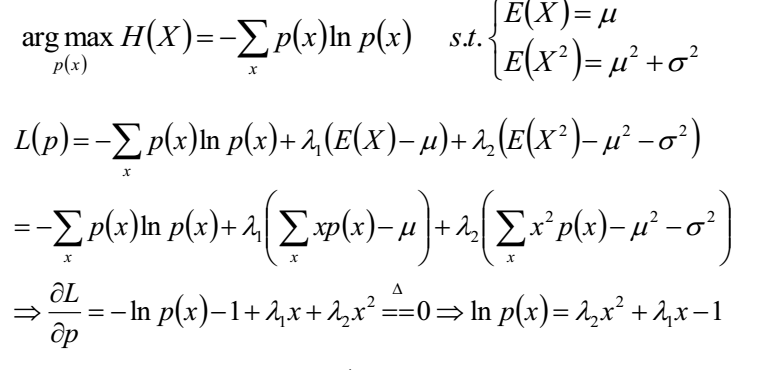
建立目标函数：



使用方差公式化简方差：

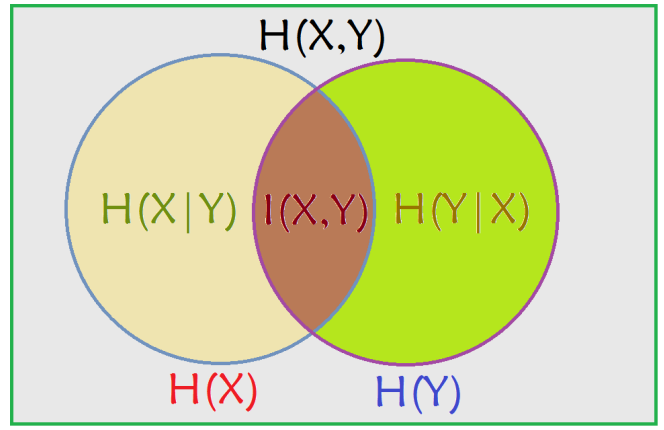


带约束的极值问题使用拉格朗日乘子法：



根据上面的分析，P(x)的对数关于随机变量x的二次形式，所以，该分布p(x)必然是正态分布。

## Venn图理解几个熵的概念

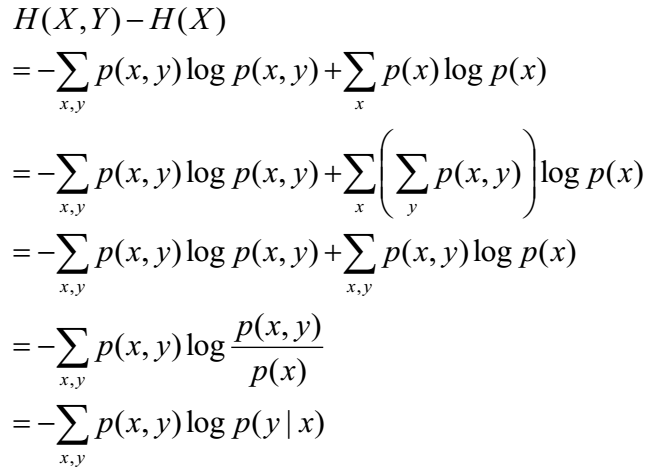


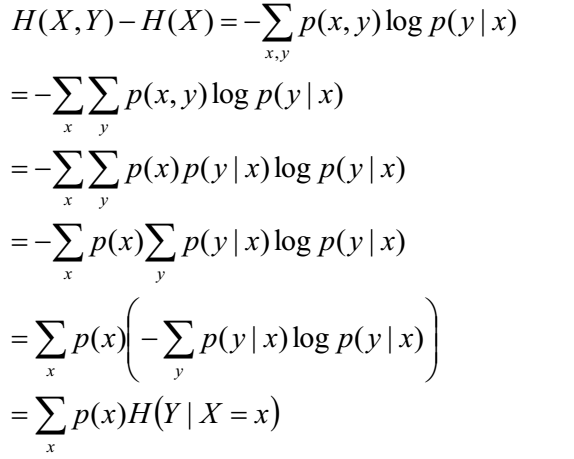
## 条件熵

 条件熵的定义

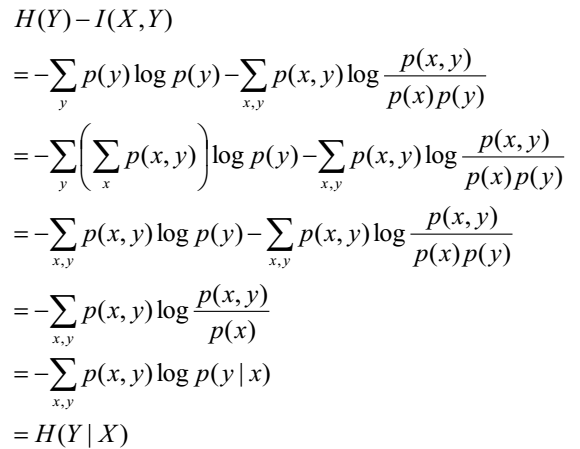
 根据互信息得到条件熵

两个随机变量X,Y的联合分布，可以形成联合熵H(X,Y)表示，（X,Y）发生所包含的熵，减去X单独发生的熵，就是在X发生前提下Y发生带来的熵即H(X|Y)=H（X,Y）-H（X）



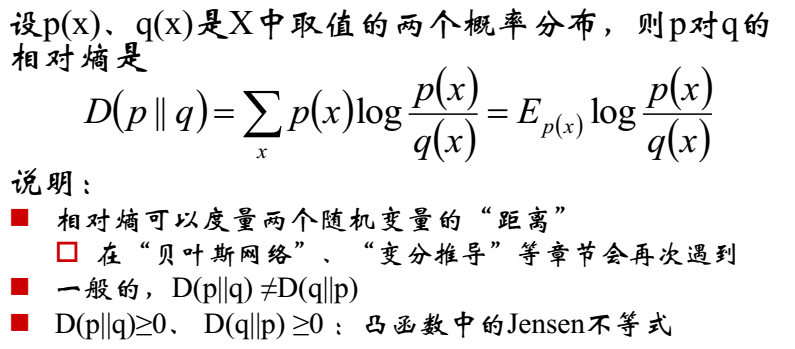


利用互信息得到条件熵



## 交叉熵（相对熵，KL散度）

设p(x),q(x)是X中取值的两个概率分布，则p对q的相对熵是



* 交叉熵可以度量两个随机变量之间的”距离“
* 
* 

互信息



