《机器学习实战》学习笔记（三）：决策树

原创 我是管小亮 2019-08-22 09:28:48 3168 收藏 24 版权
分类专栏： Machine Learning♡ 文章标签： 机器学习 机器学习实战 读书笔记 决策树

欢迎关注WX公众号：【程序员管小亮】

【机器学习】《机器学习实战》读书笔记及代码 总目录

- https://blog.csdn.net/TeFuirnever/article/details/99701256

GitHub代码地址：

- https://github.com/TeFuirnever/Machine-Learning-in-Action

目录

本章内容

- 决策树简介
- 在数据集中度量一致性
- 使用递归构造决策树
- 使用Matplotlib绘制树形图

关于决策树的西瓜书笔记和课后习题的博客如下：

- 读书笔记 https://blog.csdn.net/TeFuirnever/article/details/98944801
- 习题参考答案 https://blog.csdn.net/TeFuirnever/article/details/99056057

1、决策树

你是否玩过二十个问题的游戏，游戏的规则很简单：参与游戏的一方在脑海里想某个事物，其他参与者向他提问题，只允许提20个问题，问题的答案也只能用对或错回答。问问题的人通过推断分解，逐步缩小待猜测事物的范围。或者是酒桌上的猜数游戏，游戏规则有点类似，参与游戏的一方在脑海里想一个固定的数值，需要在固定的范围内，其他参与者进行猜测，他会先给出猜测的正确与否，如果正确直接喝酒，如果错误，就在下一个参与者开始之前给出新的猜数范围，逐步缩小待猜测事物的范围。

决策树的工作原理与上面两个游戏类似，用户输入一系列数据，然后给出游戏的答案。我们经常使用决策树处理分类问题，近来的调查表明决策树也是最经常使用的数据挖掘算法。它之所以如此流行，一个很重要的原因就是不需要了解机器学习的知识，就能搞明白决策树是如何工作的。

如果以前没有接触过决策树，也完全不用担心，它的概念非常简单。通过简单的图形就可以了解其工作原理，图3-1所示的流程图就是一个决策树，长方形代表 **判断模块**（decision block），椭圆形代表 **终止模块**（terminating block），表示已经得出结论，可以终止运行。从 **判断模块** 引出的左右箭头称作 **分支**（branch），它可以到达另一个判断模块或者终止模块。

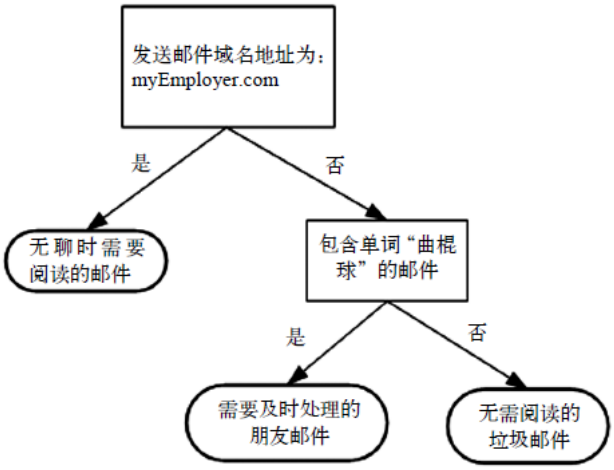


图3-1　流程图形式的决策树

上面构造的是一个假想的邮件分类系统， 首先它检测发送邮件域名地址：

- 如果地址为myEmployer.com，则将其放在分类“无聊时需要阅读的邮件”中。
- 如果邮件不是来自这个域名，则检查邮件内容里是否包含单词曲棍球。
  - 如果包含则将邮件归类到“需要及时处理的朋友邮件”。
  - 如果不包含则将邮件归类到“无需阅读的垃圾邮件”。

第二章介绍的k-近邻算法可以完成很多分类任务，详见博客：【机器学习】《机器学习实战》读书笔记及代码：第2章 - k-近邻算法，但是它最大的缺点就是无法给出数据的内在含义，决策树的主要优势就在于数据形式非常容易理解，因此决策树可以使用不熟悉的数据集合，并从中提取出一系列规则，在这些机器根据数据集创建规则时，就是 **机器学习** 的过程。

> 决策树给出结果往往可以匹敌在当前领域具有几十年工作经验的人类专家。（有点厉害0-0）

那么如何从一堆原始数据中构造决策树呢？别着急，过程如下：

- 首先讨论构造决策树的方法，以及如何编写构造树的Python代码；
- 接着提出一些度量算法成功率的方法；
- 最后使用递归建立分类器，并且使用Matplotlib绘制决策树图。

> 决策树
> 优点：计算复杂度不高，输出结果易于理解，对中间值的缺失不敏感，可以处理不相关特征数据。
> 缺点：可能会产生过度匹配问题。
> 适用数据类型：数值型和标称型。

2、决策树的构造

> 决策树的一般流程
> (1) 收集数据：可以使用任何方法。
> (2) 准备数据：树构造算法只适用于标称型数据，因此数值型数据必须离散化。
> (3) 分析数据：可以使用任何方法，构造树完成之后，我们应该检查图形是否符合预期。
> (4) 训练算法：构造树的数据结构。
> (5) 测试算法：使用经验树计算错误率。
> (6) 使用算法：此步骤可以适用于任何监督学习算法，而使用决策树可以更好地理解数据的内在含义。

决策树要如何构建呢？通常，这一过程可以概括为3个步骤：特征选择、决策树的生成和决策树的修剪。

1）特征选择

特征选择在于选取对训练数据具有分类能力的特征，这样可以提高决策树学习的效率；如果利用一个特征进行分类的结果与随机分类的结果没有很大差别，则称这个特征是没有分类能力的。经验上扔掉这样的特征对决策树学习的精度影响不大。

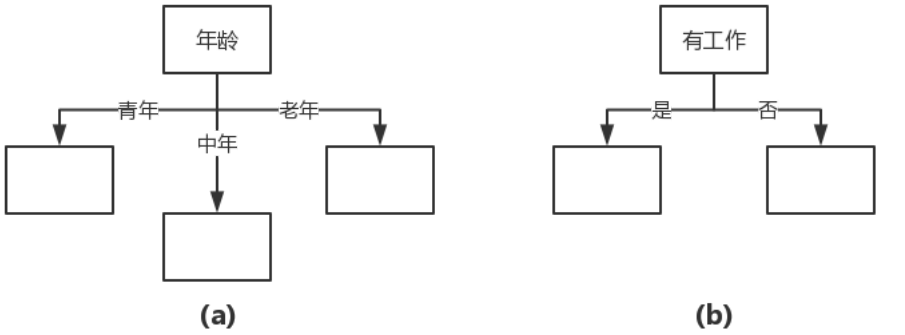
通常特征选择的标准是信息增益(information gain)或信息增益比，为了简单，本文章使用信息增益作为选择特征的标准。那么，什么是信息增益?

在继续讲解之前，先看一组实例，贷款申请样本数据表。

ID	年龄
1	青年
2	青年
3	青年
4	青年
5	青年
6	中年
7	中年
8	中年
9	中年
10	中年
11	老年
12	老年
13	老年
14	老年
15	老年

希望通过所给的训练数据学习一个贷款申请的决策树，用以对未来的贷款申请进行分类，即当新的客户提出贷款申请时，根据申请人的特征利用决策树决定是否批准贷款申请。

特征选择就是决定用哪个特征来划分特征空间。比如，我们通过上述数据表得到两个可能的决策树，分别由两个不同特征的根结点构成。



现在我们想要决定选择第一个特征还是第二个特征，但是问题是：究竟选择哪个特征更好些？这就要求确定选择特征的准则。直观上，如果一个特征具有更好的分类能力，或者说，按照这一特征将训练数据集分割成子集，使得各个子集在当前条件下有最好的分类，那么就更应该选择这个特征。信息增益就能够很好地表示这一直观的准则。那么什么是信息增益呢?

在划分数据集之前之后信息发生的变化成为信息增益，知道如何计算信息增益，我们就可以计算每个特征值划分数据集获得的信息增益，获得信息增益最高的特征就是最好的选择。

1. 香农熵

在划分数据集之前之后信息发生的变化称为 **信息增益**，知道如何计算信息增益，就可以计算每个特征值划分数据集获得的 **信息增益**，获得 **信息增益** 最高的特征就是最好的选择。在可以评测哪种数据划分方式是最好的数据划分之前，我们必须学习如何计算 **信息增益。**集合信息的度量方式称为 **香农熵** 或者简称为 **熵**，这个名字来源于信息论之父克劳德·香农。

如果看不明白什么是信息增益（information gain）和熵（entropy），请不要着急——它们自诞生的那一天起，就注定会令世人十分费解。克劳德·香农写完信息论之后，约翰·冯·诺依曼建议使用“熵”这个术语，因为大家都不知道它是什么意思。

熵定义为信息的期望值，在明晰这个概念之前，我们必须知道信息的定义。如果待分类的事务可能划分在多个分类之中，则符号 x_i 的信息定义为

$$l(x_i) = -\log_2 p(x_i)$$

其中 $p(x_i)$ 是选择该分类的概率，就是个固定公式，记住就ok了。

为了计算熵，我们需要计算所有类别，所有可能值包含的信息期望值，通过下面的公式得到：

$$H = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i)$$

其中 n 是分类的数目。

当熵中的概率由数据估计(特别是最大似然估计)得到时，所对应的熵称为 **经验熵**(empirical entropy)。定义贷款申请样本数据表中的数据为训练数据集D，则训练数据集D的经验熵为H(D)，|D|表示其样本容量，及样本个数。设有K个类 C_k，k = 1,2,3,⋯,K，$|C_k|$ 为属于类 C_k 的样本个数，这经验熵公式可以写为：

$$\mathrm{H(D)} = -\sum_{k=1}^{K} \frac{|c_k|}{|D|} \log_2 \frac{|C_k|}{|D|}$$

根据此公式计算经验熵H(D)，分析贷款申请样本数据表中的数据。最终分类结果只有两类，即放贷和不放贷。根据表中的数据统计可知，在15个数据中，9个数据的结果为放贷，6个数据的结果为不放贷。所以数据集D的经验熵H(D)为：

$$\mathrm{H(D)} = -\frac{9}{15}\log_2\frac{9}{15} - \frac{6}{15}\log_2\frac{6}{15} = 0.971$$

经过计算可知，数据集D的经验熵H(D)的值为0.971。

下面就可以准备编写代码来进行验证，在这之前，先对数据集进行属性标注。

- 年龄：0代表青年，1代表中年，2代表老年；
- 有工作：0代表否，1代表是；
- 有自己的房子：0代表否，1代表是；
- 信贷情况：0代表一般，1代表好，2代表非常好；
- 类别(是否给贷款)：no代表否，yes代表是。

```
from math import log

"""
Parameters:
    无
Returns:
    dataSet - 数据集
    labels - 分类属性
"""
# 函数说明:创建测试数据集
def createDataSet():
    dataSet = [[0, 0, 0, 0, 'no'],#数据集
            [0, 0, 0, 1, 'no'],
            [0, 1, 0, 1, 'yes'],
            [0, 1, 1, 0, 'yes'],
            [0, 0, 0, 0, 'no'],
            [1, 0, 0, 0, 'no'],
            [1, 0, 0, 1, 'no'],
            [1, 1, 1, 1, 'yes'],
            [1, 0, 1, 2, 'yes'],
            [1, 0, 1, 2, 'yes'],
            [2, 0, 1, 2, 'yes'],
            [2, 0, 1, 1, 'yes'],
            [2, 1, 0, 1, 'yes'],
```

```
            [2, 1, 0, 2, 'yes'],
            [2, 0, 0, 0, 'no']]
    labels = ['年龄', '有工作', '有自己的房子', '信贷情况']#分类属性
    return dataSet, labels#返回数据集和分类属性

"""
Parameters:
    dataSet - 数据集
Returns:
    shannonEnt - 经验熵(香农熵)
"""
# 函数说明:计算给定数据集的经验熵(香农熵)
def calcShannonEnt(dataSet):
    numEntires = len(dataSet)                        #返回数据集的行数
    labelCounts = {}                                 #保存每个标签(Label)出现次数的字典
    for featVec in dataSet:                          #对每组特征向量进行统计
        currentLabel = featVec[-1]                   #提取标签(Label)信息
        if currentLabel not in labelCounts.keys():   #如果标签(Label)没有放入统计次数的字典,添加进去
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1               #Label计数
    shannonEnt = 0.0                                 #经验熵(香农熵)
    for key in labelCounts:                          #计算香农熵
        prob = float(labelCounts[key]) / numEntires #选择该标签(Label)的概率
        shannonEnt -= prob * log(prob, 2)            #利用公式计算
    return shannonEnt                                #返回经验熵(香农熵)


if __name__ == '__main__':
    dataSet, features = createDataSet()
    print(dataSet)
    print(calcShannonEnt(dataSet))
```

```
>>>
[[0, 0, 0, 0, 'no'], [0, 0, 0, 1, 'no'], [0, 1, 0, 1, 'yes'], [0, 1, 1, 0, 'yes'], [0, 0, 0, 0, 'no'], [1, 0, 0, 0, 'no'], [1, 0
0.9709505944546686
```

2. 信息增益

如何选择特征，需要看 **信息增益**。也就是说，**信息增益** 是相对于特征而言的，**信息增益** 越大，特征对最终的分类结果影响也就越大，所以我们应该选择对最终分类结果影响最大的那个特征作为分类特征。在讲解信息增益定义之前，还需要明确一个概念，**条件熵**。熵我们知道是什么，**条件熵** 又是个什么鬼？**条件熵** H(Y|X) 表示在已知随机变量X的条件下随机变量Y的不确定性，随机变量X给定的条件下随机变量Y的 **条件熵**(conditional entropy) H(Y|X)，定义X在给定Y条件下的条件概率分布的熵对X的数学期望：

$$\mathrm{H(Y|X)} = \sum_{i=1}^{n} p_i H(Y|X = x_i)$$

这里，$p_i = P(X = x_i)$，$i = 1$，2，。。。n。同理，当 **条件熵** 中的概率由数据估计(特别是极大似然估计)得到时，所对应的 **条件熵** 成为 **条件经验熵**(empirical conditional entropy)。

接下来说说 **信息增益**，前面也提到了，**信息增益** 是相对于特征而言的。所以，特征A对训练数据集D的信息增益g(D,A)，定义为集合D的经验熵H(D)与特征A给定条件下D的经验条件熵H(D|A)之差，即

$$\mathrm{g(D,A)} = H(D) - H(D|A)$$

一般地，熵H(D)与条件熵H(D|A)之差称为 **互信息**(mutual information)。决策树学习中的 **信息增益** 等价于训练数据集中类与特征的 **互信息**。

设特征A有n个不同的取值 $a_1, a_2, \cdots, a_n$，根据特征A的取值将D划分为n个子集 D_1, D_2，$\cdots, D_n$，$|D_i|$ 为 D_i 的样本个数。记子集 D_i 中属于 C_k 的样本的集合为 D_{ik}，即 $D_{ik} = D_i \cap C_k$，$|D_{ik}|$ 为 D_{ik} 的样本个数。于是经验条件熵的公式可以写为：

$$H(D|A)=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}\log_2\frac{|D_{ik}|}{|D_i|}$$

啥，听不懂，哈哈，正常，下面我们看例子说话，回头你再看概念就懂了。还是以贷款申请样本数据表为例，年龄这一行为特征 A_1，一共三个类别，分别是：青年、中年和老年。

从数据表格中可以看出年龄是青年的数据在训练数据集出现的概率是5 / 15，同理，年龄是中年和老年的数据在训练数据集出现的概率也都是5 / 15。年龄是青年的数据的最终得到贷款的概率为2 / 5，因为在五个数据中，只有两个数据显示拿到了最终的贷款，同理，年龄是中年和老年的数据最终得到贷款的概率分别为3 / 5、4 / 5。

所以计算年龄的信息增益，过程如下：

$$\begin{aligned}g(D,A_1)&=H(D)-\left[\frac{5}{15}H(D_1)+\frac{5}{15}H(D_2)+\frac{5}{15}H(D_3)\right]\\&=0.971\\&-\left[\frac{5}{15}\left(-\frac{2}{5}\log_2\frac{2}{5}-\frac{3}{5}\log_2\frac{3}{5}\right)+\frac{5}{15}\left(-\frac{3}{5}\log_2\frac{3}{5}-\frac{2}{5}\log_2\frac{2}{5}\right)\right.\\&\left.+\frac{5}{15}\left(-\frac{4}{5}\log_2\frac{4}{5}-\frac{1}{5}\log_2\frac{1}{5}\right)\right]=0.971-0.888=0.083\end{aligned}$$

同理，计算其余特征的信息增益g(D,A2)、g(D,A3)和g(D,A4)。分别为：

$$\begin{aligned}g(D,A_2)&=H(D)-\left[\frac{5}{15}H(D_1)+\frac{15}{15}H(D_2)\right]\\&=0.971-\left[\frac{5}{15}\times 0+\frac{10}{15}\left(-\frac{4}{10}\log_2\frac{4}{10}-\frac{6}{10}\log_2\frac{6}{10}\right)\right]\\&=0.971-0.647=0.324\end{aligned}$$

$$\begin{aligned}g(D,A_3)&=H(D)-\left[\frac{6}{15}H(D_1)+\frac{9}{15}H(D_2)\right]\\&=0.971-\left[\frac{6}{15}\times 0+\frac{9}{15}\left(-\frac{3}{9}\log_2\frac{3}{9}-\frac{6}{9}\log_2\frac{6}{9}\right)\right]\\&=0.971-0.551=0.420\end{aligned}$$

$$g(D,A_4)=0.971-0.608=0.363$$

最后，比较特征的信息增益大小，发现特征A3(有自己的房子)的信息增益值最大，所以选择A3作为最优特征。

```python
from math import log

"""
Parameters:
    dataSet - 数据集
Returns:
    shannonEnt - 经验熵(香农熵)
"""
# 函数说明:计算给定数据集的经验熵(香农熵)
def calcShannonEnt(dataSet):
    numEntires = len(dataSet)                        #返回数据集的行数
    labelCounts = {}                                 #保存每个标签(Label)出现次数的字典
    for featVec in dataSet:                          #对每组特征向量进行统计
        currentLabel = featVec[-1]                   #提取标签(Label)信息
        if currentLabel not in labelCounts.keys():   #如果标签(Label)没有放入统计次数的字典,添加进去
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1               #Label计数

```

```
    shannonEnt = 0.0                                #经验熵(香农熵)
    for key in labelCounts:                         #计算香农熵
        prob = float(labelCounts[key]) / numEntires #选择该标签(Label)的概率
        shannonEnt -= prob * log(prob, 2)           #利用公式计算
    return shannonEnt                               #返回经验熵(香农熵)

"""
Parameters:
    无
Returns:
    dataSet - 数据集
    labels - 分类属性
"""
# 函数说明:创建测试数据集
def createDataSet():
    dataSet = [[0, 0, 0, 0, 'no'],#数据集
            [0, 0, 0, 1, 'no'],
            [0, 1, 0, 1, 'yes'],
            [0, 1, 1, 0, 'yes'],
            [0, 0, 0, 0, 'no'],
            [1, 0, 0, 0, 'no'],
            [1, 0, 0, 1, 'no'],
            [1, 1, 1, 1, 'yes'],
            [1, 0, 1, 2, 'yes'],
            [1, 0, 1, 2, 'yes'],
            [2, 0, 1, 2, 'yes'],
            [2, 0, 1, 1, 'yes'],
            [2, 1, 0, 1, 'yes'],
            [2, 1, 0, 2, 'yes'],
            [2, 0, 0, 0, 'no']]
    labels = ['年龄', '有工作', '有自己的房子', '信贷情况']#分类属性
    return dataSet, labels#返回数据集和分类属性

"""
Parameters:
    dataSet - 待划分的数据集
    axis - 划分数据集的特征
    value - 需要返回的特征的值
Returns:
    无
"""
# 函数说明:按照给定特征划分数据集
def splitDataSet(dataSet, axis, value):
    retDataSet = []                                 #创建返回的数据集列表
    for featVec in dataSet:                         #遍历数据集
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]         #去掉axis特征
            reducedFeatVec.extend(featVec[axis+1:])#将符合条件的添加到返回的数据集
            retDataSet.append(reducedFeatVec)
    return retDataSet                               #返回划分后的数据集

"""
Parameters:
    dataSet - 数据集
Returns:
    bestFeature - 信息增益最大的(最优)特征的索引值
"""
# 函数说明:选择最优特征
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1                   #特征数量
    baseEntropy = calcShannonEnt(dataSet)               #计算数据集的香农熵
    bestInfoGain = 0.0                                  #信息增益
    bestFeature = -1                                    #最优特征的索引值
    for i in range(numFeatures):                        #遍历所有特征
        #获取dataSet的第i个所有特征
        featList = [example[i] for example in dataSet]
        uniqueVals = set(featList)                       #创建set集合{},元素不可重复
        newEntropy = 0.0                                 #经验条件熵
        for value in uniqueVals:                         #计算信息增益
            subDataSet = splitDataSet(dataSet, i, value)   #subDataSet划分后的子集
            prob = len(subDataSet) / float(len(dataSet))   #计算子集的概率
            newEntropy += prob * calcShannonEnt(subDataSet)#根据公式计算经验条件熵
        infoGain = baseEntropy - newEntropy                 #信息增益
        print("第%d个特征的增益为%.3f" % (i, infoGain))      #打印每个特征的信息增益
        if (infoGain > bestInfoGain):                       #计算信息增益
            #更新信息增益,找到最大的信息增益
```

```
            bestInfoGain = infoGain                         #更新信息增益，找到最大的信息增益
            bestFeature = i                                 #记录信息增益最大的特征的索引值
    return bestFeature                                      #返回信息增益最大的特征的索引值


if __name__ == '__main__':
    dataSet, features = createDataSet()
    print("最优特征索引值:" + str(chooseBestFeatureToSplit(dataSet)))
```

```
>>>
第0个特征的增益为0.083
第1个特征的增益为0.324
第2个特征的增益为0.420
第3个特征的增益为0.363
最优特征索引值:2
```

最佳索引是2，也就是特征A3(有自己的房子)。

2）决策树的生成和修剪

ID3算法的核心是在决策树各个结点上对应 **信息增益** 准则选择特征，递归地构建决策树，ID3相当于用极大似然法进行概率模型的选择。

> 具体方法是：
> 从根结点(root node)开始，对结点计算所有可能的特征的信息增益，选择信息增益最大的特征作为结点的特征，由该特征的不同取值建立子节点；
> 再对子结点递归地调用以上方法，构建决策树；
> 直到所有特征的信息增益均很小或没有特征可以选择为止，最后得到一个决策树。

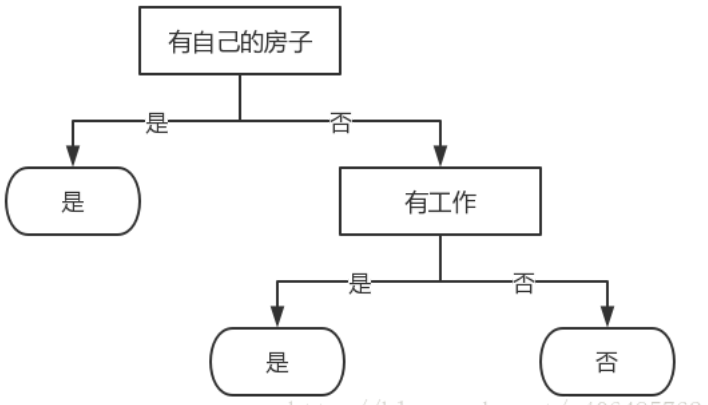
上面得到了特征A3(有自己的房子)的信息增益值最大，所以选择特征A3作为根结点的特征。它将训练集D划分为两个子集D1(A3取值为”是”)和D2(A3取值为”否”)。由于D1只有同一类的样本点，所以它成为一个叶结点，结点的类标记为“是”。对D2则需要从特征A1(年龄)，A2(有工作)和A4(信贷情况)中选择新的特征，计算各个特征的信息增益：

$$g(D2,A1) = H(D2) - H(D2 \mid A1) = 0.251$$

$$g(D2,A2) = H(D2) - H(D2 \mid A2) = 0.918$$

$$g(D2,A3) = H(D2) - H(D2 \mid A3) = 0.474$$

根据计算，选择信息增益最大的特征A2(有工作)作为结点的特征。由于A2有两个可能取值，从这一结点引出两个子结点：一个对应”是”(有工作)的子结点，包含3个样本，它们属于同一类，所以这是一个叶结点，类标记为”是”；另一个是对应”否”(无工作)的子结点，包含6个样本，它们也属于同一类，所以这也是一个叶结点，类标记为”否”。这样就生成了一个决策树，该决策树只用了两个特征(有两个内部结点)，生成的决策树如下图所示：




```
from math import log
import operator

"""
Parameters:
```

```
Parameters:
    dataSet - 数据集
Returns:
    shannonEnt - 经验熵(香农熵)
"""
# 函数说明:计算给定数据集的经验熵(香农熵)
def calcShannonEnt(dataSet):
    numEntires = len(dataSet)                        #返回数据集的行数
    labelCounts = {}                                 #保存每个标签(Label)出现次数的字典
    for featVec in dataSet:                          #对每组特征向量进行统计
        currentLabel = featVec[-1]                   #提取标签(Label)信息
        if currentLabel not in labelCounts.keys(): #如果标签(Label)没有放入统计次数的字典,添加进去
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1               #Label计数
    shannonEnt = 0.0                                 #经验熵(香农熵)
    for key in labelCounts:                          #计算香农熵
        prob = float(labelCounts[key]) / numEntires#选择该标签(Label)的概率
        shannonEnt -= prob * log(prob, 2)            #利用公式计算
    return shannonEnt                                #返回经验熵(香农熵)

"""
Parameters:
    无
Returns:
    dataSet - 数据集
    labels - 特征标签
"""
# 函数说明:创建测试数据集
def createDataSet():
    dataSet = [[0, 0, 0, 0, 'no'],#数据集
            [0, 0, 0, 1, 'no'],
            [0, 1, 0, 1, 'yes'],
            [0, 1, 1, 0, 'yes'],
            [0, 0, 0, 0, 'no'],
            [1, 0, 0, 0, 'no'],
            [1, 0, 0, 1, 'no'],
            [1, 1, 1, 1, 'yes'],
            [1, 0, 1, 2, 'yes'],
            [1, 0, 1, 2, 'yes'],
            [2, 0, 1, 2, 'yes'],
            [2, 0, 1, 1, 'yes'],
            [2, 1, 0, 1, 'yes'],
            [2, 1, 0, 2, 'yes'],
            [2, 0, 0, 0, 'no']]
    labels = ['年龄', '有工作', '有自己的房子', '信贷情况']#特征标签
    return dataSet, labels#返回数据集和分类属性

"""
Parameters:
    dataSet - 待划分的数据集
    axis - 划分数据集的特征
    value - 需要返回的特征的值
Returns:
    无
"""
# 函数说明:按照给定特征划分数据集
def splitDataSet(dataSet, axis, value):
    retDataSet = []                                  #创建返回的数据集列表
    for featVec in dataSet:                          #遍历数据集
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]          #去掉axis特征
            reducedFeatVec.extend(featVec[axis+1:])#将符合条件的添加到返回的数据集
            retDataSet.append(reducedFeatVec)
    return retDataSet                                #返回划分后的数据集

"""
Parameters:
    dataSet - 数据集
Returns:
    bestFeature - 信息增益最大的(最优)特征的索引值
"""
# 函数说明:选择最优特征
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1                    #特征数量
    baseEntropy = calcShannonEnt(dataSet)                #计算数据集的香农熵
    bestInfoGain = 0.0                                   #信息增益
    bestFeature = -1                                     #最优特征的索引值
```

```
    bestFeature = -1                                    #最优特征的索引值
    for i in range(numFeatures):                        #遍历所有特征
        #获取dataSet的第i个所有特征
        featList = [example[i] for example in dataSet]
        uniqueVals = set(featList)                      #创建set集合{},元素不可重复
        newEntropy = 0.0                                #经验条件熵
        for value in uniqueVals:                        #计算信息增益
            subDataSet = splitDataSet(dataSet, i, value)   #subDataSet划分后的子集
            prob = len(subDataSet) / float(len(dataSet))   #计算子集的概率
            newEntropy += prob * calcShannonEnt(subDataSet)#根据公式计算经验条件熵
        infoGain = baseEntropy - newEntropy             #信息增益
        # print("第%d个特征的增益为%.3f" % (i, infoGain))   #打印每个特征的信息增益
        if (infoGain > bestInfoGain):                   #计算信息增益
            bestInfoGain = infoGain                     #更新信息增益，找到最大的信息增益
            bestFeature = i                             #记录信息增益最大的特征的索引值
    return bestFeature                                  #返回信息增益最大的特征的索引值

"""
Parameters:
    classList - 类标签列表
Returns:
    sortedClassCount[0][0] - 出现此处最多的元素(类标签)
"""
# 函数说明:统计classList中出现此处最多的元素(类标签)
def majorityCnt(classList):
    classCount = {}
    for vote in classList:#统计classList中每个元素出现的次数
        if vote not in classCount.keys():classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.items(), key = operator.itemgetter(1), reverse = True)#根据字典的值降序排序
    return sortedClassCount[0][0]#返回classList中出现次数最多的元素

"""
Parameters:
    dataSet - 训练数据集
    labels - 分类属性标签
    featLabels - 存储选择的最优特征标签
Returns:
    myTree - 决策树
"""
# 函数说明:创建决策树
def createTree(dataSet, labels, featLabels):
    classList = [example[-1] for example in dataSet]       #取分类标签(是否放贷:yes or no)
    if classList.count(classList[0]) == len(classList):    #如果类别完全相同则停止继续划分
        return classList[0]
    if len(dataSet[0]) == 1:                               #遍历完所有特征时返回出现次数最多的类标签
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)           #选择最优特征
    bestFeatLabel = labels[bestFeat]                       #最优特征的标签
    featLabels.append(bestFeatLabel)
    myTree = {bestFeatLabel:{}}                            #根据最优特征的标签生成树
    del(labels[bestFeat])                                  #删除已经使用特征标签
    featValues = [example[bestFeat] for example in dataSet]#得到训练集中所有最优特征的属性值
    uniqueVals = set(featValues)                           #去掉重复的属性值
    for value in uniqueVals:                               #遍历特征，创建决策树。
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), labels, featLabels)
    return myTree


if __name__ == '__main__':
    dataSet, labels = createDataSet()
    featLabels = []
    myTree = createTree(dataSet, labels, featLabels)
    print(myTree)
```

```
>>>
{'有自己的房子': {0: {'有工作': {0: 'no', 1: 'yes'}}, 1: 'yes'}}
```

可见，决策树构建完成了。很明显这个决策树看着别扭，虽然能看懂，但是如果多点的结点，就不一定了。

下面我们使用强大的Matplotlib绘制决策树。

3、决策树的可视化

可视化需要用到的函数：

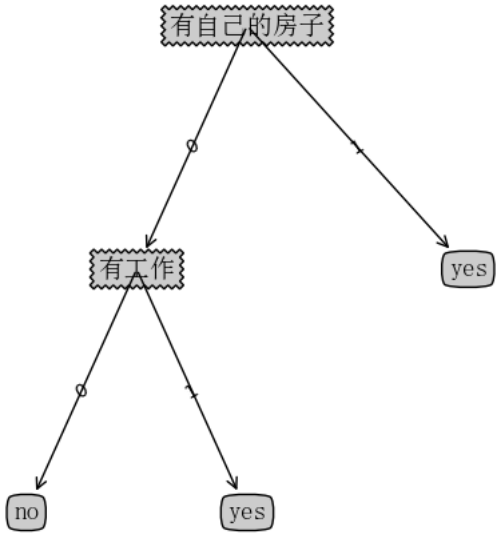
- getNumLeafs：获取决策树叶子结点的数目
- getTreeDepth：获取决策树的层数
- plotNode：绘制结点
- plotMidText：标注有向边属性值
- plotTree：绘制决策树
- createPlot：创建绘制面板
- 为了显示中文，需要设置FontProperties

```
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt
from math import log
import operator

"""
Parameters:
    dataSet - 数据集
Returns:
    shannonEnt - 经验熵(香农熵)
"""
# 函数说明:计算给定数据集的经验熵(香农熵)
def calcShannonEnt(dataSet):
    numEntires = len(dataSet)                        #返回数据集的行数
    labelCounts = {}                                 #保存每个标签(Label)出现次数的字典
    for featVec in dataSet:                          #对每组特征向量进行统计
        currentLabel = featVec[-1]                   #提取标签(Label)信息
        if currentLabel not in labelCounts.keys(): #如果标签(Label)没有放入统计次数的字典,添加进去
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1               #Label计数
    shannonEnt = 0.0                                 #经验熵(香农熵)
    for key in labelCounts:                          #计算香农熵
        prob = float(labelCounts[key]) / numEntires#选择该标签(Label)的概率
        shannonEnt -= prob * log(prob, 2)            #利用公式计算
    return shannonEnt                                #返回经验熵(香农熵)

"""
Parameters:
    无
Returns:
    dataSet - 数据集
    labels - 特征标签
"""
# 函数说明:创建测试数据集
def createDataSet():
    dataSet = [[0, 0, 0, 0, 'no'],#数据集
               [0, 0, 0, 1, 'no'],
               [0, 1, 0, 1, 'yes'],
               [0, 1, 1, 0, 'yes'],
               [0, 0, 0, 0, 'no'],
               [1, 0, 0, 0, 'no'],
               [1, 0, 0, 1, 'no'],
               [1, 1, 1, 1, 'yes'],
               [1, 0, 1, 2, 'yes'],
               [1, 0, 1, 2, 'yes'],
               [2, 0, 1, 2, 'yes'],
               [2, 0, 1, 1, 'yes'],
               [2, 1, 0, 1, 'yes'],
               [2, 1, 0, 2, 'yes'],
               [2, 0, 0, 0, 'no']]
    labels = ['年龄', '有工作', '有自己的房子', '信贷情况']#特征标签
    return dataSet, labels#返回数据集和分类属性

"""
Parameters:
    dataSet - 待划分的数据集
    axis - 划分数据集的特征
    value - 需要返回的特征的值
Returns:
```

```
    无
"""
# 函数说明:按照给定特征划分数据集
def splitDataSet(dataSet, axis, value):
    retDataSet = []                                     #创建返回的数据集列表
    for featVec in dataSet:                             #遍历数据集
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]             #去掉axis特征
            reducedFeatVec.extend(featVec[axis+1:])#将符合条件的添加到返回的数据集
            retDataSet.append(reducedFeatVec)
    return retDataSet                                   #返回划分后的数据集

"""
Parameters:
    dataSet - 数据集
Returns:
    bestFeature - 信息增益最大的(最优)特征的索引值
"""
# 函数说明:选择最优特征
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1                   #特征数量
    baseEntropy = calcShannonEnt(dataSet)               #计算数据集的香农熵
    bestInfoGain = 0.0                                  #信息增益
    bestFeature = -1                                    #最优特征的索引值
    for i in range(numFeatures):                        #遍历所有特征
        #获取dataSet的第i个所有特征
        featList = [example[i] for example in dataSet]
        uniqueVals = set(featList)                       #创建set集合{},元素不可重复
        newEntropy = 0.0                                 #经验条件熵
        for value in uniqueVals:                         #计算信息增益
            subDataSet = splitDataSet(dataSet, i, value)   #subDataSet划分后的子集
            prob = len(subDataSet) / float(len(dataSet))   #计算子集的概率
            newEntropy += prob * calcShannonEnt(subDataSet)#根据公式计算经验条件熵
        infoGain = baseEntropy - newEntropy                 #信息增益
        # print("第%d个特征的增益为%.3f" % (i, infoGain))   #打印每个特征的信息增益
        if (infoGain > bestInfoGain):                       #计算信息增益
            bestInfoGain = infoGain                         #更新信息增益，找到最大的信息增益
            bestFeature = i                                 #记录信息增益最大的特征的索引值
    return bestFeature                                      #返回信息增益最大的特征的索引值

"""
Parameters:
    classList - 类标签列表
Returns:
    sortedClassCount[0][0] - 出现此处最多的元素(类标签)
"""
# 函数说明:统计classList中出现此处最多的元素(类标签)
def majorityCnt(classList):
    classCount = {}
    for vote in classList:#统计classList中每个元素出现的次数
        if vote not in classCount.keys():classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.items(), key = operator.itemgetter(1), reverse = True)#根据字典的值降序排序
    return sortedClassCount[0][0]#返回classList中出现次数最多的元素

"""
Parameters:
    dataSet - 训练数据集
    labels - 分类属性标签
    featLabels - 存储选择的最优特征标签
Returns:
    myTree - 决策树
"""
# 函数说明:创建决策树
def createTree(dataSet, labels, featLabels):
    classList = [example[-1] for example in dataSet]        #取分类标签(是否放贷:yes or no)
    if classList.count(classList[0]) == len(classList):     #如果类别完全相同则停止继续划分
        return classList[0]
    if len(dataSet[0]) == 1:                                #遍历完所有特征时返回出现次数最多的类标签
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)            #选择最优特征
    bestFeatLabel = labels[bestFeat]                        #最优特征的标签
    featLabels.append(bestFeatLabel)
    myTree = {bestFeatLabel:{}}                             #根据最优特征的标签生成树
    del(labels[bestFeat])                                   #删除已经使用特征标签
```

```
    featValues = [example[bestFeat] for example in dataSet]#得到训练集中所有最优特征的属性值
    uniqueVals = set(featValues)                            #去掉重复的属性值
    for value in uniqueVals:                                #遍历特征，创建决策树。
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), labels, featLabels)
    return myTree

"""
Parameters:
    myTree - 决策树
Returns:
    numLeafs - 决策树的叶子结点的数目
"""
# 函数说明:获取决策树叶子结点的数目
def getNumLeafs(myTree):
    numLeafs = 0                                    #初始化叶子
    #python3中myTree.keys()返回的是dict_keys,不在是list,
    #所以不能使用myTree.keys()[0]的方法获取结点属性，可以使用list(myTree.keys())[0]
    firstStr = next(iter(myTree))
    secondDict = myTree[firstStr]                   #获取下一组字典
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':#测试该结点是否为字典，如果不是字典，代表此结点为叶子结点
            numLeafs += getNumLeafs(secondDict[key])
        else:   numLeafs +=1
    return numLeafs

"""
Parameters:
    myTree - 决策树
Returns:
    maxDepth - 决策树的层数
"""
# 函数说明:获取决策树的层数
def getTreeDepth(myTree):
    maxDepth = 0                                    #初始化决策树深度
    #python3中myTree.keys()返回的是dict_keys,不在是list,
    #所以不能使用myTree.keys()[0]的方法获取结点属性，可以使用list(myTree.keys())[0]
    firstStr = next(iter(myTree))
    secondDict = myTree[firstStr]                   #获取下一个字典
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':        #测试该结点是否为字典，如果不是字典，代表此结点为叶子结点
            thisDepth = 1 + getTreeDepth(secondDict[key])
        else:   thisDepth = 1
        if thisDepth > maxDepth: maxDepth = thisDepth     #更新层数
    return maxDepth

"""
Parameters:
    nodeTxt - 结点名
    centerPt - 文本位置
    parentPt - 标注的箭头位置
    nodeType - 结点格式
Returns:
    无
"""
# 函数说明:绘制结点
def plotNode(nodeTxt, centerPt, parentPt, nodeType):
    arrow_args = dict(arrowstyle="<-")                                      #定义箭头格式
    font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc", size=14)    #设置中文字体
    createPlot.ax1.annotate(nodeTxt, xy=parentPt,  xycoords='axes fraction',#绘制结点
        xytext=centerPt, textcoords='axes fraction',
        va="center", ha="center", bbox=nodeType, arrowprops=arrow_args, FontProperties=font)

"""
Parameters:
    cntrPt、parentPt - 用于计算标注位置
    txtString - 标注的内容
Returns:
    无
"""
# 函数说明:标注有向边属性值
def plotMidText(cntrPt, parentPt, txtString):
    xMid = (parentPt[0]-cntrPt[0])/2.0 + cntrPt[0]#计算标注位置
    yMid = (parentPt[1]-cntrPt[1])/2.0 + cntrPt[1]
    createPlot.ax1.text(xMid, yMid, txtString, va="center", ha="center", rotation=30)

"""
Parameters:
```

```
    myTree - 决策树(字典)
    parentPt - 标注的内容
    nodeTxt - 结点名
Returns:
    无
"""
# 函数说明:绘制决策树
def plotTree(myTree, parentPt, nodeTxt):
    decisionNode = dict(boxstyle="sawtooth", fc="0.8")                                      #设置结点格式
    leafNode = dict(boxstyle="round4", fc="0.8")                                            #设置叶结点格式
    numLeafs = getNumLeafs(myTree)                                                          #获取决策树叶结点数目,

    depth = getTreeDepth(myTree)                                                            #获取决策树层数
    firstStr = next(iter(myTree))                                                           #下个字典
    cntrPt = (plotTree.xOff + (1.0 + float(numLeafs))/2.0/plotTree.totalW, plotTree.yOff)#中心位置
    plotMidText(cntrPt, parentPt, nodeTxt)                                                  #标注有向边属性值
    plotNode(firstStr, cntrPt, parentPt, decisionNode)                                      #绘制结点
    secondDict = myTree[firstStr]                                                           #下一个字典,

    plotTree.yOff = plotTree.yOff - 1.0/plotTree.totalD                                     #y偏移
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':                                          #测试该结点是否为字典,


            plotTree(secondDict[key],cntrPt,str(key))                                       #不是叶结点,

        else:                                                                               #如果是叶结点,绘制叶结点,

            plotTree.xOff = plotTree.xOff + 1.0/plotTree.totalW
            plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff), cntrPt, leafNode)
            plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))
    plotTree.yOff = plotTree.yOff + 1.0/plotTree.totalD

"""
Parameters:
    inTree - 决策树(字典)
Returns:
    无
"""
# 函数说明:创建绘制面板
def createPlot(inTree):
    fig = plt.figure(1, facecolor='white')                  #创建fig
    fig.clf()                                               #清空fig
    axprops = dict(xticks=[], yticks=[])
    createPlot.ax1 = plt.subplot(111, frameon=False, **axprops)#去掉x、y轴
    plotTree.totalW = float(getNumLeafs(inTree))            #获取决策树叶结点数目
    plotTree.totalD = float(getTreeDepth(inTree))           #获取决策树层数
    plotTree.xOff = -0.5/plotTree.totalW; plotTree.yOff = 1.0; #x偏移
    plotTree(inTree, (0.5,1.0), '')                         #绘制决策树
    plt.show()                                              #显示绘制结果


if __name__ == '__main__':
    dataSet, labels = createDataSet()
    featLabels = []
    myTree = createTree(dataSet, labels, featLabels)
    print(myTree)
    createPlot(myTree)
```



4、使用决策树执行分类

依靠训练数据构造了决策树之后，可以将它用于实际数据的分类。在执行数据分类时，需要决策树以及用于构造树的标签向量；然后，程序比较测试数据与决策树上的数值，递归执行该过程直到进入叶子结点；最后将测试数据定义为叶子结点所属的类型。

在构建决策树的代码，可以看到，有个 `featLabels` 参数。它是用来记录各个分类结点的，在用决策树做预测的时候，按顺序输入需要的分类结点的属性值即可。举个例子，比如用上述已经训练好的决策树做分类，只需要提供这个人是否有房子，是否有工作这两个信息即可，无需提供其他冗余的信息。

```
# -*- coding: UTF-8 -*-
from math import log
import operator

"""
Parameters:
    dataSet - 数据集
Returns:
    shannonEnt - 经验熵(香农熵)
"""
# 函数说明:计算给定数据集的经验熵(香农熵)
def calcShannonEnt(dataSet):
    numEntires = len(dataSet)                        #返回数据集的行数
    labelCounts = {}                                 #保存每个标签(Label)出现次数的字典
    for featVec in dataSet:                          #对每组特征向量进行统计
        currentLabel = featVec[-1]                   #提取标签(Label)信息
        if currentLabel not in labelCounts.keys(): #如果标签(Label)没有放入统计次数的字典,添加进去
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1               #Label计数
    shannonEnt = 0.0                                 #经验熵(香农熵)
    for key in labelCounts:                          #计算香农熵
        prob = float(labelCounts[key]) / numEntires#选择该标签(Label)的概率
        shannonEnt -= prob * log(prob, 2)            #利用公式计算
    return shannonEnt                                #返回经验熵(香农熵)

"""
Parameters:
    无
Returns:
    dataSet - 数据集
    labels - 特征标签
"""
# 函数说明:创建测试数据集
def createDataSet():
    dataSet = [[0, 0, 0, 0, 'no'],#数据集
            [0, 0, 0, 1, 'no'],
            [0, 1, 0, 1, 'yes'],
            [0, 1, 1, 0, 'yes'],

```

```
            [0, 0, 0, 0, 'no'],
            [1, 0, 0, 0, 'no'],
            [1, 0, 0, 1, 'no'],
            [1, 1, 1, 1, 'yes'],
            [1, 0, 1, 2, 'yes'],
            [1, 0, 1, 2, 'yes'],
            [2, 0, 1, 2, 'yes'],
            [2, 0, 1, 1, 'yes'],
            [2, 1, 0, 1, 'yes'],
            [2, 1, 0, 2, 'yes'],
            [2, 0, 0, 0, 'no']]
    labels = ['年龄', '有工作', '有自己的房子', '信贷情况']#特征标签
    return dataSet, labels#返回数据集和分类属性

"""
Parameters:
    dataSet - 待划分的数据集
    axis - 划分数据集的特征
    value - 需要返回的特征的值
Returns:
    无
"""
# 函数说明:按照给定特征划分数据集
def splitDataSet(dataSet, axis, value):
    retDataSet = []                                     #创建返回的数据集列表
    for featVec in dataSet:                             #遍历数据集
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]             #去掉axis特征
            reducedFeatVec.extend(featVec[axis+1:])#将符合条件的添加到返回的数据集
            retDataSet.append(reducedFeatVec)
    return retDataSet                                   #返回划分后的数据集

"""
Parameters:
    dataSet - 数据集
Returns:
    bestFeature - 信息增益最大的(最优)特征的索引值
"""
# 函数说明:选择最优特征
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1                   #特征数量
    baseEntropy = calcShannonEnt(dataSet)               #计算数据集的香农熵
    bestInfoGain = 0.0                                  #信息增益
    bestFeature = -1                                    #最优特征的索引值
    for i in range(numFeatures):                        #遍历所有特征
        #获取dataSet的第i个所有特征
        featList = [example[i] for example in dataSet]
        uniqueVals = set(featList)                      #创建set集合{},元素不可重复
        newEntropy = 0.0                                #经验条件熵
        for value in uniqueVals:                        #计算信息增益
            subDataSet = splitDataSet(dataSet, i, value)   #subDataSet划分后的子集
            prob = len(subDataSet) / float(len(dataSet))   #计算子集的概率
            newEntropy += prob * calcShannonEnt(subDataSet)#根据公式计算经验条件熵
        infoGain = baseEntropy - newEntropy             #信息增益
        # print("第%d个特征的增益为%.3f" % (i, infoGain))   #打印每个特征的信息增益
        if (infoGain > bestInfoGain):                   #计算信息增益
            bestInfoGain = infoGain                     #更新信息增益，找到最大的信息增益
            bestFeature = i                             #记录信息增益最大的特征的索引值
    return bestFeature                                  #返回信息增益最大的特征的索引值

"""
Parameters:
    classList - 类标签列表
Returns:
    sortedClassCount[0][0] - 出现此处最多的元素(类标签)
"""
# 函数说明:统计classList中出现此处最多的元素(类标签)
def majorityCnt(classList):
    classCount = {}
    for vote in classList:                                        #统计classList中每个元素出现的次数
        if vote not in classCount.keys():classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.items(), key = operator.itemgetter(1), reverse = True)#根据字典的值降序排序
    return sortedClassCount[0][0]                                 #返回classList中出现次数最多的元素

"""
```

```
    Parameters:
        dataSet - 训练数据集
        labels - 分类属性标签
        featLabels - 存储选择的最优特征标签
    Returns:
        myTree - 决策树
    """
    # 函数说明:创建决策树
    def createTree(dataSet, labels, featLabels):
        classList = [example[-1] for example in dataSet]        #取分类标签(是否放贷:yes or no)
        if classList.count(classList[0]) == len(classList):    #如果类别完全相同则停止继续划分
            return classList[0]
        if len(dataSet[0]) == 1:                                #遍历完所有特征时返回出现次数最多的类标签
            return majorityCnt(classList)
        bestFeat = chooseBestFeatureToSplit(dataSet)            #选择最优特征
        bestFeatLabel = labels[bestFeat]                        #最优特征的标签
        featLabels.append(bestFeatLabel)
        myTree = {bestFeatLabel:{}}                             #根据最优特征的标签生成树
        del(labels[bestFeat])                                   #删除已经使用特征标签
        featValues = [example[bestFeat] for example in dataSet]#得到训练集中所有最优特征的属性值
        uniqueVals = set(featValues)                            #去掉重复的属性值
        for value in uniqueVals:                                #遍历特征，创建决策树。
            myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), labels, featLabels)
        return myTree

    """
    Parameters:
        inputTree - 已经生成的决策树
        featLabels - 存储选择的最优特征标签
        testVec - 测试数据列表，顺序对应最优特征标签
    Returns:
        classLabel - 分类结果
    """
    # 函数说明:使用决策树分类
    def classify(inputTree, featLabels, testVec):
        firstStr = next(iter(inputTree))      #获取决策树结点
        secondDict = inputTree[firstStr]      #下一个字典
        featIndex = featLabels.index(firstStr)
        for key in secondDict.keys():
            if testVec[featIndex] == key:
                if type(secondDict[key]).__name__ == 'dict':
                    classLabel = classify(secondDict[key], featLabels, testVec)
                else: classLabel = secondDict[key]
        return classLabel


    if __name__ == '__main__':
        dataSet, labels = createDataSet()
        featLabels = []
        myTree = createTree(dataSet, labels, featLabels)
        testVec = [0,1]#测试数据
        result = classify(myTree, featLabels, testVec)
        if result == 'yes':
            print('放贷')
        if result == 'no':
            print('不放贷')
```

这里只增加了 `classify` 函数，用于决策树分类。输入测试数据[0,1]，代表没有房子，但是有工作，分类结果如下所示：

```
>>>
放贷
```

5、决策树的存储

构造决策树是很耗时的任务，即使处理很小的数据集，如前面的样本数据，也要花费几秒的时间，如果数据集很大，将会耗费很多计算时间。然而用创建好的决策树解决分类问题，则可以很快完成。因此，为了节省计算时间，最好能够在每次执行分类时调用已经构造好的决策树。

为了解决这个问题，需要使用Python模块 `pickle` 序列化对象。序列化对象可以在磁盘上保存对象，并在需要的时候读取出来。假设我们已经得到决策树{'有自己的房子': {0: {'有工作': {0: 'no', 1: 'yes'}}, 1: 'yes'}}，使用 `pickle.dump` 存储决策树。

```
import pickle

"""
Parameters:
    inputTree - 已经生成的决策树
    filename - 决策树的存储文件名
Returns:
    无
"""
# 函数说明:存储决策树
def storeTree(inputTree, filename):
    with open(filename, 'wb') as fw:
        pickle.dump(inputTree, fw)


if __name__ == '__main__':
    myTree = {'有自己的房子': {0: {'有工作': {0: 'no', 1: 'yes'}}, 1: 'yes'}}
    storeTree(myTree, 'classifierStorage.txt')
```

运行代码，在该Python文件的相同目录下，会生成一个名为classifierStorage.txt的txt文件，这个是个二进制存储的文件，会生成即可，下次使用的时候：

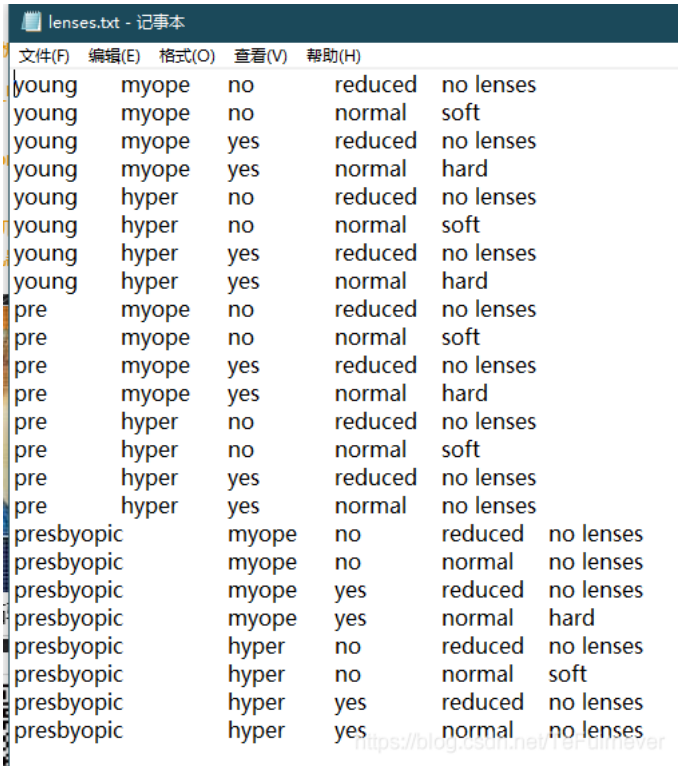
```
import pickle

"""
Parameters:
    filename - 决策树的存储文件名
Returns:
    pickle.load(fr) - 决策树字典
"""
# 函数说明:读取决策树
def grabTree(filename):
    fr = open(filename, 'rb')
    return pickle.load(fr)


if __name__ == '__main__':
    myTree = grabTree('classifierStorage.txt')
    print(myTree)
```

```
>>>
{'有自己的房子': {0: {'有工作': {0: 'no', 1: 'yes'}}, 1: 'yes'}}
```

6、Sklearn构建决策树分类器预测隐形眼镜类型

1. 准备数据

一共有24组数据，数据的Labels依次是age、prescript、astigmatic、tearRate、class，也就是第一列是年龄，第二列是症状，第三列是是否散光，第四列是眼泪数量，第五列是最终的分类标签。



2. 测试算法

对string类型的数据序列化，需要先生成pandas数据，这样方便序列化工作。所以先原始数据->字典->pandas数据，编写代码如下：

```
import pandas as pd

if __name__ == '__main__':
    with open('lenses.txt', 'r') as fr:                                 #加载文件
        lenses = [inst.strip().split('\t') for inst in fr.readlines()]#处理文件
    lenses_target = []                                                  #提取每组数据的类别，保存在列表里
    for each in lenses:
        lenses_target.append(each[-1])

    lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']    #特征标签
    lenses_list = []                                                #保存Lenses数据的临时列表
    lenses_dict = {}                                                #保存Lenses数据的字典，用于生成pandas
    for each_label in lensesLabels:                                 #提取信息，生成字典
        for each in lenses:
            lenses_list.append(each[lensesLabels.index(each_label)])
        lenses_dict[each_label] = lenses_list
        lenses_list = []
    print(lenses_dict)                                              #打印字典信息
    lenses_pd = pd.DataFrame(lenses_dict)                           #生成pandas.DataFrame
    print(lenses_pd)
```

```
>>>
{'age': ['young', 'young', 'young', 'young', 'young', 'young', 'young', 'young', 'pre', 'pre', 'pre', 'pre', 'pre', 'pre', 'pre'
          age astigmatic prescript tearRate
0       young         no     myope  reduced
1       young         no     myope   normal
2       young        yes     myope  reduced
3       young        yes     myope   normal
4       young         no     hyper  reduced
5       young         no     hyper   normal
6       young        yes     hyper  reduced
7       young        yes     hyper   normal
8         pre         no     myope  reduced
9         pre         no     myope   normal
10        pre        yes     myope  reduced
11        pre        yes     myope   normal
```

```
12        pre         no     hyper  reduced
13        pre         no     hyper   normal
14        pre        yes     hyper  reduced
15        pre        yes     hyper   normal
16  presbyopic         no     myope  reduced
17  presbyopic         no     myope   normal
18  presbyopic        yes     myope  reduced
19  presbyopic        yes     myope   normal
20  presbyopic         no     hyper  reduced
21  presbyopic         no     hyper   normal
22  presbyopic        yes     hyper  reduced
23  presbyopic        yes     hyper   normal
```

接下来，将数据序列化，编写代码如下：

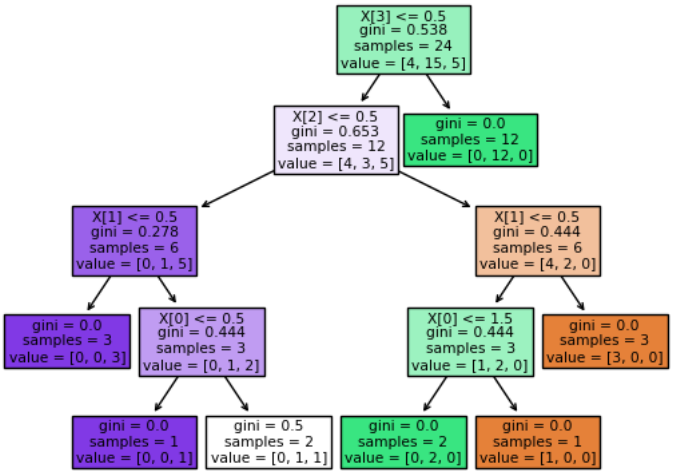
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

if __name__ == '__main__':
    with open('lenses.txt', 'r') as fr:                                 #加载文件
        lenses = [inst.strip().split('\t') for inst in fr.readlines()]#处理文件
    lenses_target = []                                                  #提取每组数据的类别，保存在列表里
    for each in lenses:
        lenses_target.append(each[-1])

    lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']       #特征标签
    lenses_list = []                                                    #保存Lenses数据的临时列表
    lenses_dict = {}                                                    #保存Lenses数据的字典，用于生成pandas
    for each_label in lensesLabels:                                     #提取信息，生成字典
        for each in lenses:
            lenses_list.append(each[lensesLabels.index(each_label)])
        lenses_dict[each_label] = lenses_list
        lenses_list = []
    # print(lenses_dict)                                                #打印字典信息
    lenses_pd = pd.DataFrame(lenses_dict)                               #生成pandas.DataFrame
    print(lenses_pd)                                                    #打印pandas.DataFrame
    le = LabelEncoder()                                                 #创建LabelEncoder()对象，用于序列化
    for col in lenses_pd.columns:                                       #为每一列序列化
        lenses_pd[col] = le.fit_transform(lenses_pd[col])
    print(lenses_pd)
```

```
>>>
          age astigmatic prescript tearRate
0       young         no     myope  reduced
1       young         no     myope   normal
2       young        yes     myope  reduced
3       young        yes     myope   normal
4       young         no     hyper  reduced
5       young         no     hyper   normal
6       young        yes     hyper  reduced
7       young        yes     hyper   normal
8         pre         no     myope  reduced
9         pre         no     myope   normal
10        pre        yes     myope  reduced
11        pre        yes     myope   normal
12        pre         no     hyper  reduced
13        pre         no     hyper   normal
14        pre        yes     hyper  reduced
15        pre        yes     hyper   normal
16  presbyopic         no     myope  reduced
17  presbyopic         no     myope   normal
18  presbyopic        yes     myope  reduced
19  presbyopic        yes     myope   normal
20  presbyopic         no     hyper  reduced
21  presbyopic         no     hyper   normal
22  presbyopic        yes     hyper  reduced
23  presbyopic        yes     hyper   normal
```

```
    age  astigmatic  prescript  tearRate
0     2           0          1         1
1     2           0          1         0
2     2           1          1         1
3     2           1          1         0
4     2           0          0         1
5     2           0          0         0
6     2           1          0         1
7     2           1          0         0
8     0           0          1         1
9     0           0          1         0
10    0           1          1         1
11    0           1          1         0
12    0           0          0         1
13    0           0          0         0
14    0           1          0         1
15    0           1          0         0
16    1           0          1         1
17    1           0          1         0
18    1           1          1         1
19    1           1          1         0
20    1           0          0         1
21    1           0          0         0
22    1           1          0         1
23    1           1          0         0
```

可以看出数据已经顺利完成了序列化，可以进行fit()数据了。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.externals.six import StringIO
from sklearn import tree

if __name__ == '__main__':
    with open('lenses.txt', 'r') as fr:                                 #加载文件
        lenses = [inst.strip().split('\t') for inst in fr.readlines()]#处理文件
    lenses_target = []                                                  #提取每组数据的类别，保存在列表里
    for each in lenses:
        lenses_target.append(each[-1])
    print(lenses_target)

    lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']     #特征标签
    lenses_list = []                                                    #保存Lenses数据的临时列表
    lenses_dict = {}                                                    #保存Lenses数据的字典，用于生成pandas
    for each_label in lensesLabels:                                     #提取信息，生成字典
        for each in lenses:
            lenses_list.append(each[lensesLabels.index(each_label)])
        lenses_dict[each_label] = lenses_list
        lenses_list = []
    # print(lenses_dict)                                                #打印字典信息
    lenses_pd = pd.DataFrame(lenses_dict)                               #生成pandas.DataFrame
    # print(lenses_pd)                                                  #打印pandas.DataFrame
    le = LabelEncoder()                                                 #创建LabelEncoder()对象，用于序列化
    for col in lenses_pd.columns:                                       #序列化
        lenses_pd[col] = le.fit_transform(lenses_pd[col])
    # print(lenses_pd)                                                  #打印编码信息

    clf = tree.DecisionTreeClassifier(max_depth = 4)                    #创建DecisionTreeClassifier()类
    # clf = clf.fit(lenses_pd.values.tolist(), lenses_target)           #使用数据，构建决策树
    tree.plot_tree(clf.fit(lenses_pd.values.tolist(), lenses_target), filled=True)
    plt.show()
```



官方可视化代码示例教程：https://scikit-learn.org/stable/auto_examples/tree/plot_iris_dtc.html

```
print(clf.predict([[1,1,1,0]]))#预测
```

```
>>>
['hard']
```

7、sklearn.tree.DecisionTreeClassifier

sklearn.tree.DecisionTreeClassifier是一个很好的模型，决策树算法就是通过它实现的，详细的看这个博客——sklearn.tree.DecisionTreeClassifier()函数解析（最清晰的解释）

8、总结

决策树分类器就像带有终止块的流程图，终止块表示分类结果。开始处理数据集时，首先需要测量集合中数据的不一致性，也就是熵，然后寻找最优方案划分数据集，直到数据集中的所有数据属于同一分类。ID3算法可以用于划分标称型数据集。构建决策树时，通常采用递归的方法将数据集转化为决策树。一般并不构造新的数据结构，而是使用Python语言内嵌的
数据结构字典存储树节点信息。

使用Matplotlib的注解功能，可以将存储的树结构转化为容易理解的图形。Python语言的pickle模块可用于存储决策树的结构。隐形眼镜的例子表明决策树可能会产生过多的数据集划分，从而产生过度匹配数据集的问题。我们可以通过裁剪决策树，合并相邻的无法产生大量信息增益的叶节点，消除过度匹配问题。还有其他的决策树的构造算法，最流行的是C4.5和CART，第9章讨论回归问题时将介绍CART算法。

第2章、第3章讨论的是结果确定的分类算法，数据实例最终会被明确划分到某个分类中。

- 【机器学习】《机器学习实战》读书笔记及代码：第2章 - k-近邻算法
- 【机器学习】《机器学习实战》读书笔记及代码：第3章 - 决策树

下一章讨论的分类算法将不能完全确定数据实例应该划分到某个分类，或者只能给出数据实例属于给定分类的概率。