构建最优决策树问题

一、决策树概述

1.1 决策树的基本概念

决策树(Decision Tree)是一种树形结构，其中每个内部节点表示一个属性上的测试，每个分支代表一个测试输出，每个叶节点代表一种类别。决策树是在已知各种情况发生概率的基础上，通过构成决策树来求取净现值的期望值大于等于零的概率，评价项目风险，判断其可行性的决策分析方法，是直观运用概率分析的一种图解法。由于这种决策分支画成图形很像一棵树的枝干，故称决策树。在机器学习中，决策树是一个预测模型，他代表的是对象属性与对象值之间的一种映射关系。

机器学习中，决策树是一个预测模型；它代表的是对象属性与对象值之间的一种映射关系。树中每个节点表示某个对象，而每个分叉路径则代表的某个可能的属性值，而每个叶结点则对应从根节点到该叶节点所经历的路径所表示的对象的值。决策树仅有单一输出，若欲有复数输出，可以建立独立的决策树以处理不同输出。数据挖掘中决策树是一种经常要用到的技术，可以用于分析数据，同样也可以用来作预测。

从数据产生决策树的机器学习技术叫做决策树学习, 通俗说就是决策树。该树由决策节点、分支和叶子 节点组成。每个决策节点表示一个待分类的数 据类别或属性，每个叶子节点表示一种结果[1]。 整个决策的过程从根决策节点开始，从上到下，依次判断，直到给出分类结果。

1.2 决策树的应用

决策树最先提出来是应用在分类问题上，主要用来做分类预测，作为十大经典机器学习算法，决策树的应用十分广泛，无论是在分类问题还是回归问题，都可以看到决策树的身影，随着机器学习的不断发展与应用的越来越广泛，决策树作为基础的分类算法，也不断被改良。

在医学领域，决策树为预测一些潜在的疾病提供一种解决方法，比如采用决策树模型与 logistic 回归模型分析影响农村胃癌高危人群干预效果的影响因素[2], 初次之外，根据临床检验资料信息,利用决策树模型建立 2 型糖尿病预测模型,为能更准确地诊断 2 型糖尿病提出理论依据[3]。这些简单的案例都是比较新的应用，决策树算法在提出之后，就为医学领域的临床疾病预测提供有效的参考办法。

在公共领域，决策树可以用来分析网民情感[4]，帮助政府即使了解民众对热点事件或者最新政策的反应，有助于智慧政府的建设，对于公共出行，可以借助与机器学习来对客流量进行预测，方便交管部分及时疏流，减少交通拥堵，也可以根据根据历史交通事故信息建立决策树模型，预测事故多发地段，提醒出行群众，在社会安全领域，可以建立训练决策树模型来预测违法犯罪活动发生的地点和时间，从而更好的维护公众安全，保证社会稳定。

在灾害预测领域，决策树模型可以用来预测森林火灾等自然灾害的发生，从而提醒相关部门提前采取预防措施，降低自然灾害对社会经济以及群众生活的影响和经济损失。

基本所有的机器学习应用的场景，决策树作为机器学习的基础算法，都会或多或少的应用在其中。

1.3 常见的决策树算法

决策树算法是数据挖掘技术中一种流行算法，从其诞生到现在，很多决策树算法已经被提出，一些经典算法在实际中已经有很应用，下面是几种比较经典的算法。

1.3.1 C4.5 算法

C4.5 算法是 J.R.Quinlan 在 ID3 算法的基础上改进的一种算法，C4.5 算法继承可 ID3 算法的优点，并在下面几个方面对 ID3 算法进行了改进[5]。

（1）使用信息增益率作为选择分裂的标准，克服了用信息增益选择分裂属性时易于选择多指属性的不足。

（2）在生成树的过程中同时进行剪枝，减少了树的规模。

（3）能够处理连续型属性

（4）可以解决有数据缺失的数据集

1.3.2 CART 算法

CART(Classification And Regression Tree)采用二分递归分割的算法，因此建立的决策树都是二叉树，即树内每个节点最多只包含两个分支，树的结构简单清晰，分类规则较少[6]。

CART 的一个功能是能够选择出可以降低数据无序度的非类别属性,选择非类别属性的依据是根据它在不同预测下对样本数据划分的好坏程度。比如对于某一个非类别属性,衡量一个分裂点是否优于另一个分裂点的标准是纯度值

CART 的另一个优点是将验证模型和建立最优通用树二者结合。该算法首先生成一棵复杂的树,再根据交叉验证和测试集验证的结果对复杂树进行调整剪枝,进而得到最优的决策树。最优通用树是将剪枝后不同的树在测试数据上进行测试得到的。性能好的树在测试中的表现优秀,而复杂的树则很少能在测试数据上表现出好的性能。所以,通过使用交叉验证方法得到最适合未来数据的树。

当数据集中含有缺失值时，CART 采用替代属性来处理。当一棵决策树首选分裂属性的数据有缺失时,可以用非类别属性来替代。比如,我们将人的体重属性作为首选分类属性,但如果某个样本的体重属性缺失时,我们可以退一步,采用头发长短,这一非类别属性来作为替代属性进行分裂。

1.3.3 Fuzzy ID3 算法

ID3 算法通过使用自上而下的方法,对每一个节点进行搜寻,测试出测试属性来建立一棵完整的树。在建树过程中,ID3 算法可以应对数据噪音,因为其决策的对象是针对全部数据做出的统计性质。同时,该算法只搜索全部空间的一部分,大大降低了建树时间,提高了计算速度。另外,算法采用信息增益作为分裂属性的选择标准,通过计算,选择具有最高信息增益属性作为分裂属性[7]。

模糊 ID3 算法是对 ID3 算法的扩展。模糊 ID3 算法首先对连续属性进行模糊化过程,然后利用模糊集合的势计算模糊信息增益,从而选择分裂属性。模糊 ID3 克服了 ID3 不能处理连续属性的弱点。但是,模糊 ID3 与 ID3 相同,都不能处理缺失属性值。

1.3.4 FS-DT 算法

FS-DT 算法是 SLIQ 算法的模糊化算法，由 B.Chandra 和 P.P.Varghese 在 2008 年

提出[8]，FS-DT 算法在建树过程中的标准是寻找一个属性使其模糊 Gini Index 达到最小，在测试过程中，FS-DT 计算测试样本在每一个叶子节点中的隶属程度，并用分支权重作为这些隶程度的权重，求和之后计算此测试样本属于某一类。

FS-DT 与其他模糊决策树最大的区别在于,它不是先对连续属性进行模糊化,它对数据的模糊化过程是在建树过程中完成的,由 Gini Index 确定分裂属性及分裂点后,对数据的模糊化刁`开始。另外 FS-DT 在模糊算子的选取上选择了“乘法”算子,这有别于一般的模糊决策树选择“取小”算子。

1.3.5 Yuan’s FDT 算法

Yuan’s FDT 算法是传统决策树算法的一种推广[9]。此方法是基于模糊不确定性建立的,所以它描述的分类规则更按近人类的思考模式,具有很高的可理解性。同时，Yuan’s FDT 算法还具有很好的鲁棒性，因此可以应对数据噪音，提高适用性。

在研究问题时，Yuan’s FDT 算法可以用模糊属性和模糊语言来处理分类问题，Yuan’s FDT 算法使用分类不确定度来选择属性的方法，在构建生成树的时候某个属性使得分类不确定性达到最小，则选择其作为分类属性。

1.3.6 SPRINT 算法

SPRINT 算法是对 SLIQ 算法的一种改进[9]，其并行性更好，且更能适应大数据集，对于大数据集，SLIQ 采取类表、属性表和类直方图三种数据结构，SPRINT 算法对其进行了修改，删除其他两类结构，只留下属性列表，属性列表的每行包含属性值、属性对应的类和样本记录好三类信息，在建树过程中，随着一个内部节点分裂成两个孩子节点，属性列表页同时分裂成两个属性列表，由于连续属性的列表是预先排序的，所以分裂后生成属性列表依旧是有序的。SPRINT 算法生成的所有属性列表都可以换入或者换出内存，因此不受数据集大小的限制，但同时，由于使用了属性列表，SPRINT 算法很大程度上增加了系统存储空间负担。

1.3.7 PUBLIC 算法

PUBLIC 算法一个显著的优点就是速度快，效率高，因为该算法的建树过程和剪枝过程是一步完成的，不会由于产生将会被剪掉的枝叶而浪费时间[9]。该算法在建树的过程中和 SPRINT 算法类似，而在剪枝过程中,由于树的不完整分支蕴含的信息也是不完整的,这可能会导致误删枝叶或剪枝过度,从而影响分类准确率。针对这一问题，PUBLIC 算法采取低估策略来解决过高的代价估算，对于那些需要进一步扩展的内部节点，通过计算编码代价的较低阈值来实现，而对于叶子节点，其估算方法不变，计算较低阈值的方法是将其设置为 1。

1.3.8 CHAID 算法

CHAID 算法是一种可产生多分枝的决策树,其目标变量可以定距或定类[9]。此算法从统计显著性角度确定分裂属性及其分割点,达到优化树的分枝的目的。此算法在商业上已经得到广泛应用,例如在 SPSS 中,此算法已作为基本的决策树算法广为应用。

二、问题的转化及构建过程

2.1 决策树问题的映射

整个问题的映射可以理解为：输入的一个数据样例，假设为$X=(x_1,x_2,\dots,x_n,y)$其中$(x_1,x_2,x_3,\dots,x_n)$为每个属性都有不同的类型，可以把整个 X 认为从$x_1\dots x_n y$的一条以y作为重点的路径，而对于前面的$x_1\dots x_n$的顺序则没有限制，只是要求最后的重点必须是 y，可以将每个样例作为一个树，也可以认为是一条路径，而构造决策树的过程就是根据上面所有的路径来生成一棵树，使的所有的叶子节点都是 y，非叶子节点都是x_i，最优的决策树根据不同的评价函数来确定，可以根据树的规模大小来作为评价标准，也可以根据下面介绍的三种信息学中的度量指标来进行度量。

构建的树需要具有泛化能力，这就要求树尽可能多的覆盖输入路径，但同时也必须考虑过拟合的问题。

2.2 决策树的构建过程

决策树构建基本算法可以概括如下[10]：

输入：训练集$D=\{(x_1,y),(x_2,y),(x_3,y)\dots(x_m,y)\}$;

属性集$A=\{a_1,a_2,a_3,\dots a_d\}$;

过程：函数$TreeGenerate(D,A)$

1：生成节点 node

2：**if** D 中样本全属于同一类别 C **then**

3：将 node 标记为 C 类叶节点，return;

4：**end if**

5：**if** $A=\emptyset$ **OR** D中样本在 A 上取值相同 **then**

6：将 node 标记为叶节点，其类别标记为 D 中样本树最多的类；**return；**

7：**end if**

8：从 A 中选择最优划分属性a_*;

9：**for** a_*的每一个值a_*^v ***do***

10：为 node 生成一个分支；令D_v表示 D 中在a_*上取值为a_*^v的样本子集;

11：**if** D_v为空 **then**

12：将分支节点标为叶节点，其类别标记为 D 中样本最多的类；**return；**

13：**else**

14:以$TreeGenerate(D_V, A\setminus\{a_*\})$为分支节点

15：**end if**

16：**end for**

输出：以 node 为根节点的一颗决策树

三、三种度量决策树指标

如何选择最优划分属性，一般而言，随着划分的不断进行，我们希望决策树的分支结点所包含的样本将尽可能属于同一类别，即节点的“纯度”(purity)越来越高。

3.1 信息增益

信息熵（information entropy）在信息学中主要是用来度量样本集合的纯度，假设当前样本的集合 D 中第 k 类样本所占的比例为$\mathrm{p_k}$ $(k = 1,2, \dots n)$则 D 的信息熵定义为

$$\mathrm{Ent(D)} = -\sum_{k=1}^{n} p_k \log_2 p_k$$

Ent(D)的值越小，则 D 的纯度越高[10]。

假设离散属性 a 有 V 个可能的取值，$\{\mathrm{a}^1, \mathrm{a}^2, \dots, \mathrm{a}^\mathrm{v}\}$，若使用 a 来对样本进行划分，则会产生 V 个分割点，其中第 v 个分割点包含了 D 中所有在属性 a 上取值为a^v的样本，我们可以根据上式来计算D^v的信息熵，在考虑到不同的分支节点所包含的样本数不同，给分支节点赋予权重$|\mathrm{D}^\mathrm{v}|/|\mathrm{D}|$，即样本数越多的分支节点的影响越大，于是可以计算出用属性 a 对样本集 D 进行划分所获得的“信息增益”(information gain)

$$\mathrm{Gain(D,a)} = Ent(D) - \sum_{v=1}^{n} \frac{D^v}{D}\, Ent(D^v)$$

一般而言，信息增益越大，则意味着使用 a 来进行划分所获得的“纯度提升”越大，因此可以使用信息增益来进行决策树的划分属性选择，即在决策树的算法中的第 8 行选择属性$\mathrm{a}_* = \underbrace{argmax}_{a \in A}\ Gain(D,a)$。上述的 ID3 决策树算法就是以信息增益作为准则划分属性

3.2 信息增益率

实际上信息增益准则对可取值数目较多的属性有所偏好，为减少这种偏好可能带来的不利影响[10]，著名的 C4.5 算法并没有直接使用信息增益，而是使用“增益率”(gain ratio)来选择最优划分属性，信息增益率表示为

$$\mathrm{Gain\_ratio(D,a)} = \frac{Gain(D,a)}{IV(a)}$$

其中

$$\mathrm{IV}(a) = -\sum_{v=1}^{V} \frac{|D^v|}{|D|}\, log_2\, \frac{|D^v|}{|D|}$$

称为属性 a 的“固有值”(intrinsic value)。

3.3 基尼指数

CART 决策树使用“基尼指数”(Gini Index)来选择最优划分属性[10]，数据集 D 的纯度可以使用基尼指数来度量：

$$\mathrm{Gini(D)} = \sum_{k=1}^{n} \sum_{k' \neq k} p_k p_{k'} = 1 - \sum_{k=1}^{n} p_k^2$$

直观来说，Gini(D)反映了从数据集 D 中随机抽取两个样本，其类别标记不一致的概率，因此 Gini(D)越小，则数据集 D 的纯度越高。

属性 a 的基尼指数定义为：

$$\mathrm{Gini\_index(D,a)} = \sum_{v=1}^{n} \frac{|D^v|}{|D|} Gini(D^v)$$

于是，在候选属性集合 A 中，选择那个使得划分后基尼指数最小的属性作为最优划分属性，即$\mathrm{a}_* = \underbrace{argmax}_{a\in A} \ Gini_index(D,a)$

四、树的剪枝问题

决策树生成算法递归地产生决策树，直到不能继续下去为止，这样产生的树往往对训练数据很准确，但是对未知的测试数据的分类却没有那么准确，即出现过拟合的现象，过拟合的原因在于学习时过多地考虑如何提高对训练数据的正确分类，从而构建出过于复杂的决策树，解决这个问题的办法是考虑决策树的复杂度，从而对生成的决策树进行简化[11]。

4.1 决策树的损失函数

决策树的剪枝往往通过极小化决策树的整体损失函数(lose function)或代价函数(cost function)来实现，设树 T 的叶节点个数为|T|，t 是树 T 的叶节点，该叶节点有$\mathrm{N_t}$个样本点，其中 k 类的样本点有$\mathrm{N_{tk}}$个，$\mathrm{k} = 1, 2, \dots, \mathrm{K}$，$\mathrm{H_t}(k)$为叶节点 t 上的经验熵，$\alpha \geq 0$ 为参数，则决策树的损失函数可以定义为

$$\mathrm{C_a}(T) = \sum_{t=1}^{|T|} N_t H_t(T) + \alpha|T|$$

其中经验熵为

$$\mathrm{H_t}(T) = -\sum_{k} \frac{N_{tk}}{N_t} \log \frac{N_{tk}}{N_t}$$

在损失函数中令

$$\mathrm{C(T)} = \sum_{t=1}^{|T|} N_t H_t(T) = -\sum_{t=1}^{|T|} \sum_{k=1}^{K} N_{tk} \log \frac{N_{tk}}{N_t}$$

这时有

$$\mathrm{C_\alpha}(T) = C(T) + \alpha|T|$$

其中$\mathrm{C(T)}$表示模型对训练数据的预测误差，即模型与训练数据的拟合程度，$\mathrm{|T|}$表示模型的复杂度，参数$\alpha \geq 0$控制两者之间的影响，比较大的α促使选择较为简单的模型，较小的α促使选择较为复杂的模型，$\alpha = 0$意味着只考虑模型与训练数据集的拟合程度，不考虑模型的复杂度[11]。

4.2 决策树剪枝策略

决策树剪枝的基本策略有“预剪枝”和“后剪枝”[10]。

4.2.1 预剪枝

预剪枝是指在决策树生成过程中，对每个结点在划分前先进行估计，若当前节点

的划分不能带来决策树泛化性能的提升，则停止划分并将当前节点标记为叶节点。

对比未剪枝的决策树和经过预剪枝的决策树可以看出：预剪枝使得决策树的很多分支都没有“展开”，这不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。但是，另一方面，因为预剪枝是基于“贪心”的，所以，虽然当前划分不能提升泛华性能，但是基于该划分的后续划分却有可能导致性能提升，因此预剪枝决策树有可能带来欠拟合的风险。

4.2.2 后剪枝

后剪枝则是先从训练集生成一颗完整的决策树，然后自底向上地对非叶节点进行考察，若将该节点对应的子树替换为叶节点能带来决策树泛化性能的提高，则将该子树替换为叶节点。

对比预剪枝和后剪枝，能够发现，后剪枝决策树通常比预剪枝决策树保留了更多的分支，一般情形下，后剪枝决策树的欠拟合风险小，泛华性能往往也要优于预剪枝决策树。但后剪枝过程是在构建完全决策树之后进行的，并且要自底向上的对树中的所有非叶结点进行逐一考察，因此其训练时间开销要比未剪枝决策树和预剪枝决策树都大得多。

五、代码实现及实验测试结果

5.1 Python 代码实现

5.1.1 选择数据集

这里实现我们选择 IRIS 数据集[12]，数据集的下载地址为：http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

该数据集的每一条数据包含 5 个属性，分别是 Sepal.Length、Sepal.Width、Petal.Length、Petal.Width 以及种类。

5.1.2 代码实现

环境：Python3.6.5，IDE：PyCharm2018.2.4

实现决策树代码（这里选择信息增益作为度量指标）不适用第三方的机器学习库

```
# 对 y 的各种可能的取值出现的个数进行计数.。
# 其他函数利用该函数来计算数据集和的混杂程度
def uniquecounts(rows):
    results = {}
    for row in rows:
        #计数结果在最后一列
        r = row[len(row)-1]
        if r not in results:results[r] = 0
        results[r]+=1
    return results # 返回一个字典
# 熵
def entropy(rows):
    from math import log
```

```
    log2 = lambda x:log(x)/log(2)
    results = uniquecounts(rows)
    #开始计算熵的值
    ent = 0.0
    for r in results.keys():
        p = float(results[r])/len(rows)
        ent = ent - p*log2(p)
    return ent
#定义节点的属性
class decisionnode:
    def __init__(self,col = -1,value = None, results = None, tb = None,fb =
None):
        self.col = col   # col 是待检验的判断条件所对应的列索引值
        self.value = value # value 对应于为了使结果为 True，当前列必须匹配的值
        self.results = results #保存的是针对当前分支的结果，它是一个字典
        self.tb = tb
## desision node,对应于结果为 true 时，树上相对于当前节点的子树上的节点
        self.fb = fb
## desision node,对应于结果为 true 时，树上相对于当前节点的子树上的节点
# 基尼不纯度
# 随机放置的数据项出现于错误分类中的概率
def giniimpurity(rows):
    total = len(rows)
    counts = uniquecounts(rows)
    imp =0
    for k1 in counts:
        p1 = float(counts[k1])/total
        for k2 in counts: # 这个循环是否可以用（1-p1）替换？
            if k1 == k2: continue
            p2 = float(counts[k2])/total
            imp+=p1*p2
    return imp
# 改进 giniimpurity
def giniimpurity_2(rows):
    total = len(rows)
    counts = uniquecounts(rows)
    imp = 0
    for k1 in counts.keys():
        p1 = float(counts[k1])/total
        imp+= p1*(1-p1)
    return imp
#在某一列上对数据集进行拆分。可应用于数值型或因子型变量
def divideset(rows,column,value):
    #定义一个函数，判断当前数据行属于第一组还是第二组
    split_function = None
```

```
    if isinstance(value,int) or isinstance(value,float):
        split_function = lambda row:row[column] >= value
    else:
        split_function = lambda row:row[column]==value
    # 将数据集拆分成两个集合，并返回
    set1 = [row for row in rows if split_function(row)]
    set2 = [row for row in rows if not split_function(row)]
    return(set1,set2)
# 以递归方式构造树
def buildtree(rows,scoref = entropy):
    if len(rows)==0 : return decisionnode()
    current_score = scoref(rows)
    # 定义一些变量以记录最佳拆分条件
    best_gain = 0.0
    best_criteria = None
    best_sets = None
    column_count = len(rows[0]) - 1
    for col in range(0,column_count):
        #在当前列中生成一个由不同值构成的序列
        column_values = {}
        for row in rows:
            column_values[row[col]] = 1 # 初始化
        #根据这一列中的每个值，尝试对数据集进行拆分
        for value in column_values.keys():
            (set1,set2) = divideset(rows,col,value)
            # 信息增益
            p = float(len(set1))/len(rows)
            gain = current_score - p*scoref(set1) - (1-p)*scoref(set2)
            if gain>best_gain and len(set1)>0 and len(set2)>0:
                best_gain = gain
                best_criteria = (col,value)
                best_sets = (set1,set2)
    #创建子分支
    if best_gain>0:
        trueBranch = buildtree(best_sets[0])  #递归调用
        falseBranch = buildtree(best_sets[1])
        return decisionnode(col = best_criteria[0],value =
best_criteria[1],
                            tb = trueBranch,fb = falseBranch)
    else:
        return decisionnode(results = uniquecounts(rows))
# 决策树的显示
def printtree(tree,indent = ''):
    # 是否是叶节点
    if tree.results!=None:
```

```
        print (str(tree.results))
    else:
        # 打印判断条件
        print (str(tree.col)+":"+str(tree.value)+"? ")
        #打印分支
        print (indent+"T->",)
        printtree(tree.tb,indent+" ")
        print (indent+"F->",)
        printtree(tree.fb,indent+" ")
# 对新的观测数据进行分类
def classify(observation,tree):
    if tree.results!= None:
        return tree.results
    else:
        v = observation[tree.col]
        branch = None
        if isinstance(v,int) or isinstance(v,float):
            if v>= tree.value: branch = tree.tb
            else: branch = tree.fb
        else:
            if v==tree.value : branch = tree.tb
            else: branch = tree.fb
        return classify(observation,branch)
# 决策树的剪枝
def prune(tree,mingain):
    # 如果分支不是叶节点，则对其进行剪枝
    if tree.tb.results == None:
        prune(tree.tb,mingain)
    if tree.fb.results == None:
        prune(tree.fb,mingain)
    # 如果两个子分支都是叶节点，判断是否能够合并
    if tree.tb.results !=None and tree.fb.results !=None:
        #构造合并后的数据集
        tb,fb = [],[]
        for v,c in tree.tb.results.items():
            tb+=[[v]]*c
        for v,c in tree.fb.results.items():
            fb+=[[v]]*c
        #检查熵的减少量
        delta = entropy(tb+fb)-(entropy(tb)+entropy(fb)/2)
        if delta < mingain:
            # 合并分支
            tree.tb,tree.fb = None,None
            tree.results = uniquecounts(tb+fb)
# test
```

```
#tree = buildtree(my_data,scoref = giniimpurity)
#prune(tree,0.1)
#printtree(tree)
```

使用 sklearn 第三方机器学习库实现的决策树的算法

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
def iris_type(s):
    it = {b'Iris-setosa': 0, b'Iris-versicolor': 1, b'Iris-virginica': 2}
    return it[s]
# 花萼长度、花萼宽度，花瓣长度，花瓣宽度
# iris_feature = 'sepal length', 'sepal width', 'petal length', 'petal
width'
iris_feature = u'花萼长度', u'花萼宽度', u'花瓣长度', u'花瓣宽度'
if __name__ == "__main__":
    mpl.rcParams['font.sans-serif'] = [u'SimHei']
    mpl.rcParams['axes.unicode_minus'] = False

    path = 'E:\PycharmProjects\DecisionTreeBySklearn\iris.data'  # 数据文件路
径
    data = np.loadtxt(path, dtype=float, delimiter=',', converters={4:
iris_type})
    x, y = np.split(data, (4,), axis=1)
    # 为了可视化，仅使用前两列特征
    x = x[:, :2]
    x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=1)
    # 决策树参数估计
    # min_samples_split = 10：如果该结点包含的样本数目大于10，则(有可能)对其分支
    # min_samples_leaf = 10：若将某结点分支后，得到的每个子结点样本数目都大于10，则
##完成分支；否则，不进行分支
    #criterion 参数可以设置为 gini 即使用 Gini Index
    model = Pipeline([
        ('ss', StandardScaler()),
        ('DTC', DecisionTreeClassifier(criterion='entropy', max_depth=3))])
    # clf = DecisionTreeClassifier(criterion='entropy', max_depth=3)
    model = model.fit(x_train, y_train)
    y_test_hat = model.predict(x_test)      # 测试数据
```

```
    # 保存
    # dot -Tpng -o 1.png 1.dot
    f = open('.\iris_tree.dot', 'w')
    tree.export_graphviz(model.get_params('DTC')['DTC'], out_file=f)
    # 画图
    N, M = 100, 100  # 横纵各采样多少个值
    x1_min, x1_max = x[:, 0].min(), x[:, 0].max()  # 第 0 列的范围
    x2_min, x2_max = x[:, 1].min(), x[:, 1].max()  # 第 1 列的范围
    t1 = np.linspace(x1_min, x1_max, N)
    t2 = np.linspace(x2_min, x2_max, M)
    x1, x2 = np.meshgrid(t1, t2)  # 生成网格采样点
    x_show = np.stack((x1.flat, x2.flat), axis=1)  # 测试点
    cm_light = mpl.colors.ListedColormap(['#A0FFA0', '#FFA0A0', '#A0A0FF'])
    cm_dark = mpl.colors.ListedColormap(['g', 'r', 'b'])
    y_show_hat = model.predict(x_show)  # 预测值
    y_show_hat = y_show_hat.reshape(x1.shape)  # 使之与输入的形状相同
    plt.figure(facecolor='w')
    plt.pcolormesh(x1, x2, y_show_hat, cmap=cm_light)  # 预测值的显示
    plt.scatter(x_test[:, 0], x_test[:, 1], c=y_test.ravel(),
edgecolors='k', s=100, cmap=cm_dark, marker='o')  # 测试数据
    plt.scatter(x[:, 0], x[:, 1], c=y.ravel(), edgecolors='k', s=40,
cmap=cm_dark)  # 全部数据
    plt.xlabel(iris_feature[0], fontsize=15)
    plt.ylabel(iris_feature[1], fontsize=15)
    plt.xlim(x1_min, x1_max)
    plt.ylim(x2_min, x2_max)
    plt.grid(True)
    plt.title(u'鸢尾花数据的决策树分类', fontsize=17)
    plt.show()

    # 训练集上的预测结果
    y_test = y_test.reshape(-1)
    print(y_test_hat)
    print (y_test)
    result = (y_test_hat == y_test)   # True 则预测正确，False 则预测错误
    acc = np.mean(result)
    print ('准确度: %.2f%%' % (100 * acc))

    # 过拟合：错误率
    depth = np.arange(1, 15)
    err_list = []
    for d in depth:
        clf = DecisionTreeClassifier(criterion='entropy', max_depth=d)
        clf = clf.fit(x_train, y_train)
        y_test_hat = clf.predict(x_test)  # 测试数据
```

```
        result = (y_test_hat == y_test)  # True 则预测正确，False 则预测错误
        err = 1 - np.mean(result)
        err_list.append(err)
        print (d, ' 准确度: %.2f%%' % (100 * err))
    plt.figure(facecolor='w')
    plt.plot(depth, err_list, 'ro-', lw=2)
    plt.xlabel(u'决策树深度', fontsize=15)
    plt.ylabel(u'错误率', fontsize=15)
    plt.title(u'决策树深度与过拟合', fontsize=17)
    plt.grid(True)
    plt.show()
```

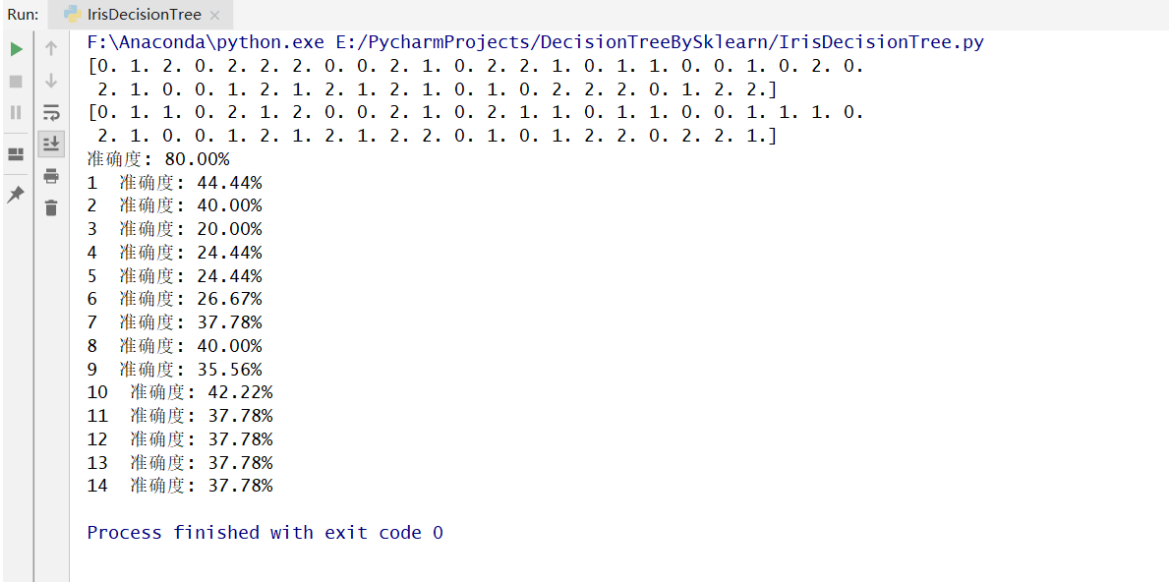
两特征组合的分类的代码

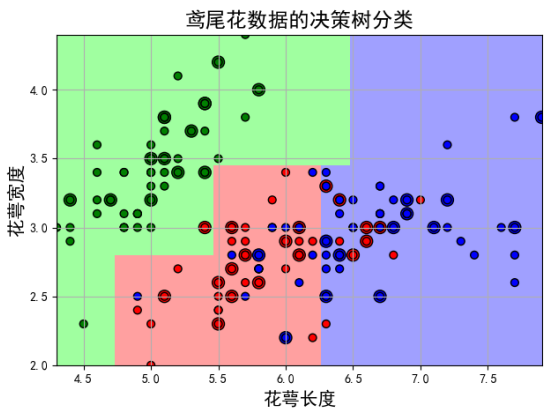
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn.tree import DecisionTreeClassifier
def iris_type(s):
    it = {b'Iris-setosa': 0, b'Iris-versicolor': 1, b'Iris-virginica': 2}
    return it[s]
# 'sepal length', 'sepal width', 'petal length', 'petal width'
iris_feature = u'花萼长度', u'花萼宽度', u'花瓣长度', u'花瓣宽度'
if __name__ == "__main__":
    mpl.rcParams['font.sans-serif'] = [u'SimHei']  # 黑体 FangSong/KaiTi
    mpl.rcParams['axes.unicode_minus'] = False
    path = 'E:\PycharmProjects\DecisionTreeBySklearn\iris.data'  # 数据文件路径
    data = np.loadtxt(path, dtype=float, delimiter=',', converters={4:
iris_type})
    x_prime, y = np.split(data, (4,), axis=1)
    feature_pairs = [[0, 1], [0, 2], [0, 3], [1, 2], [1, 3], [2, 3]]
    plt.figure(figsize=(10, 9), facecolor='#FFFFFF')
    for i, pair in enumerate(feature_pairs):
        # 准备数据
        x = x_prime[:, pair]
        # 决策树学习
        clf = DecisionTreeClassifier(criterion='entropy', min_samples_leaf=3)
        dt_clf = clf.fit(x, y)
        # 画图
        N, M = 500, 500  # 横纵各采样多少个值
        x1_min, x1_max = x[:, 0].min(), x[:, 0].max()  # 第 0 列的范围
        x2_min, x2_max = x[:, 1].min(), x[:, 1].max()  # 第 1 列的范围
        t1 = np.linspace(x1_min, x1_max, N)
        t2 = np.linspace(x2_min, x2_max, M)
        x1, x2 = np.meshgrid(t1, t2)  # 生成网格采样点
        x_test = np.stack((x1.flat, x2.flat), axis=1)  # 测试点
        # 训练集上的预测结果
```

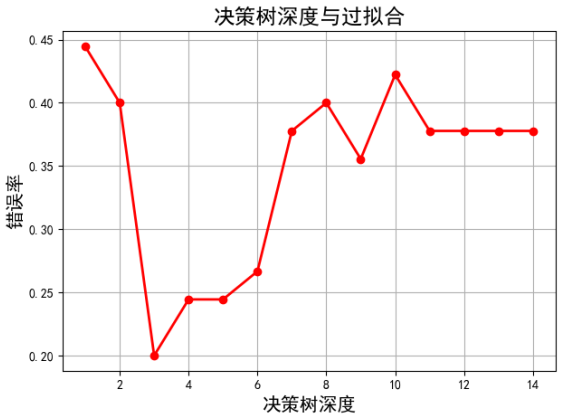
```
        y_hat = dt_clf.predict(x)
        y = y.reshape(-1)
        c = np.count_nonzero(y_hat == y)    # 统计预测正确的个数
        print ('特征：  ', iris_feature[pair[0]], ' + ', iris_feature[pair[1]],)
        print ('\t预测正确数目：', c,)
        print ('\t准确率：%.2f%%' % (100 * float(c) / float(len(y))))
        # 显示
        cm_light = mpl.colors.ListedColormap(['#A0FFA0', '#FFA0A0', '#A0A0FF'])
        cm_dark = mpl.colors.ListedColormap(['g', 'r', 'b'])
        y_hat = dt_clf.predict(x_test)  # 预测值
        y_hat = y_hat.reshape(x1.shape)  # 使之与输入的形状相同
        plt.subplot(2, 3, i+1)
        plt.pcolormesh(x1, x2, y_hat, cmap=cm_light)  # 预测值
        plt.scatter(x[:, 0], x[:, 1], c=y, edgecolors='k', cmap=cm_dark)  # 样本
        plt.xlabel(iris_feature[pair[0]], fontsize=14)
        plt.ylabel(iris_feature[pair[1]], fontsize=14)
        plt.xlim(x1_min, x1_max)
        plt.ylim(x2_min, x2_max)
        plt.grid()
    plt.suptitle(u'决策树对鸢尾花数据的两特征组合的分类结果', fontsize=18)
    plt.tight_layout(2)
    plt.subplots_adjust(top=0.92)
    plt.show()
```

5.2 实验测试结果

调用第三方 sklearn 机器学习库程序执行的结果如下所示，可以看出不做其他处理直接使用原始数据的时候准确率比较低，并且随着树的深度的加深，准确率越来越低，造成这种情况的原因是过拟合，导致准确率越来越低，并且在绘制的图中也可以看出准确率与树的深度之间的关系



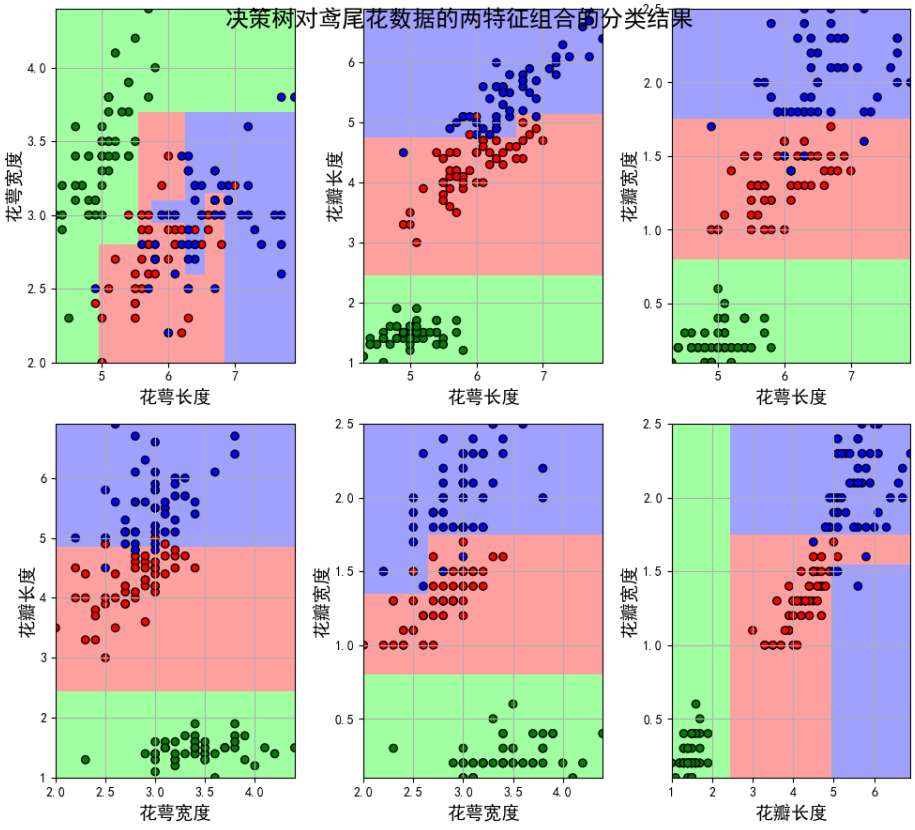




采取组合特征的方式来构建决策树，可以看出准确率明显提高。

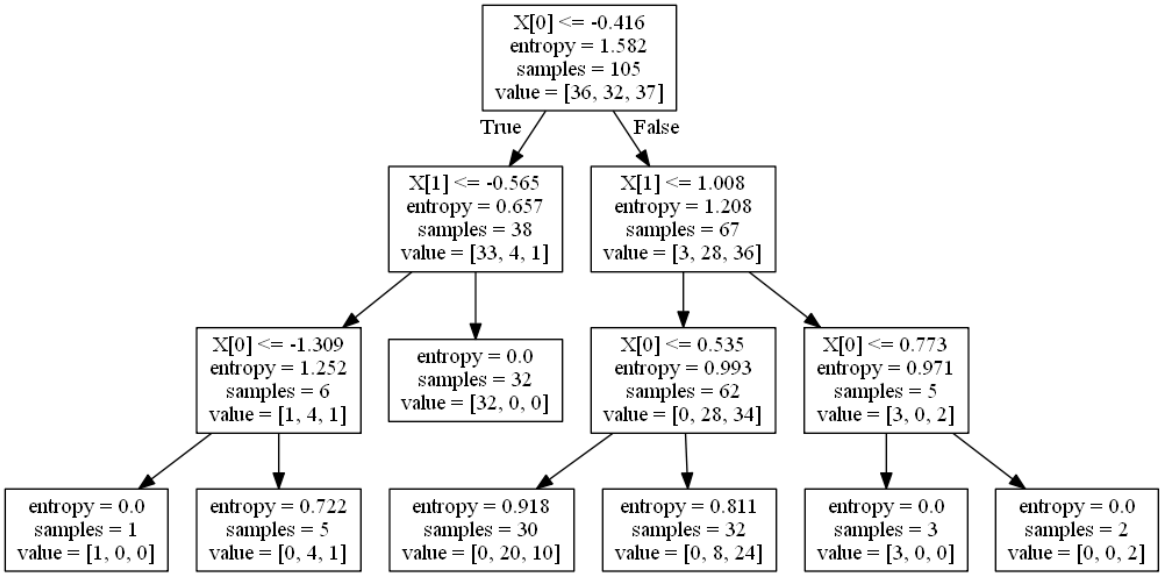
```
Run:  IrisDecisionTreeEnum
F:\Anaconda\python.exe E:/PycharmProjects/DecisionTreeBySklearn/IrisDecisionTreeEnum.py
特征:    花萼长度  +  花萼宽度
     预测正确数目:  123
     准确率: 82.00%
特征:    花萼长度  +  花瓣长度
     预测正确数目:  145
     准确率: 96.67%
特征:    花萼长度  +  花瓣宽度
     预测正确数目:  144
     准确率: 96.00%
特征:    花萼宽度  +  花瓣长度
     预测正确数目:  143
     准确率: 95.33%
特征:    花萼宽度  +  花瓣宽度
     预测正确数目:  145
     准确率: 96.67%
特征:    花瓣长度  +  花瓣宽度
     预测正确数目:  147
     准确率: 98.00%

Process finished with exit code 0
```



使用 Graphviz 命令 dot 来生成最终产生的决策树如下所示

```
dot -Tpng iris_tree.dot -o Decision.png
```



参考文献

[1] 冯亚. 数据挖掘中决策树分类算法研究与应用[D]. 西北大学, 2007.

[2] 刘兵, 李苹, 朱玫烨,等. 决策树模型与 logistic 回归模型在胃癌高危人群干预效果影响因素分析中的应用[J]. 中国卫生统计, 2018(1):70-73.

[3] 杨光, 马尔丽. 决策树模型在 2 型糖尿病预测中的应用[J]. 沈阳师范大学学报：自然科学版, 2018(3).

[4] 刘钢,张维石.基于决策树的网民评价情感分析[J].现代计算机(专业版),2017(32):15-19.

[5] Quinlan J R. C4.5: programs for machine learning[J]. 1992, 1.

[6] 张亮, 宁芊. CART 决策树的两种改进及应用[J]. 计算机工程与设计, 2015(5):1209-1213.

[7] Umanol M, Okamoto H, Hatono I, et al. Fuzzy decision trees by fuzzy ID3 algorithm and its application to diagnosis systems[C]// IEEE, International Fuzzy Systems Conference. IEEE, 1994:2113-2118 vol.3.

[8] Chandra B, Varghese P P. Fuzzy SLIQ decision tree algorithm[J]. IEEE Transactions on Systems Man & Cybernetics Part B, 2008, 38(5):1294-1301.

[9] 李旭. 五种决策树算法的比较研究[D]. 大连理工大学, 2011

[10] 周志华. 机器学习 : = Machine learning[M]. 清华大学出版社, 2016.

[11] 李航. 统计学习方法[M]. 清华大学出版社, 2012.

[12] Peter Harrington. 机器学习实战[M]. 人民邮电出版社, 2013.