第六章 决策树

决策树（decision tree）是最经典的机器学习算法之一，算法简单，计算效率高，结果易于解释，在商业、金融、医疗等领域获得了广泛应用。

1. 决策树原理

决策树的思想很朴素，从一层层的if/else问题中进行学习，并得出结论，类似于我们平时做决策的过程。例如你想要区分四种动物：熊、鹰、企鹅和海豚，你的目标是通过提出尽可能少的if/else问题来得到正确答案。你可能首先会问：这种动物有没有羽毛，这个问题会将可能的动物减少到只有两种。如果答案是“Yes”，你可再问一个问题，如会不会飞翔区分鹰和企鹅。如果是“No”，那就是海豚或熊，你再问一个问题就可以区分这两种动物，比如有没有鳍。

这个过程可以表示为一棵决策树：

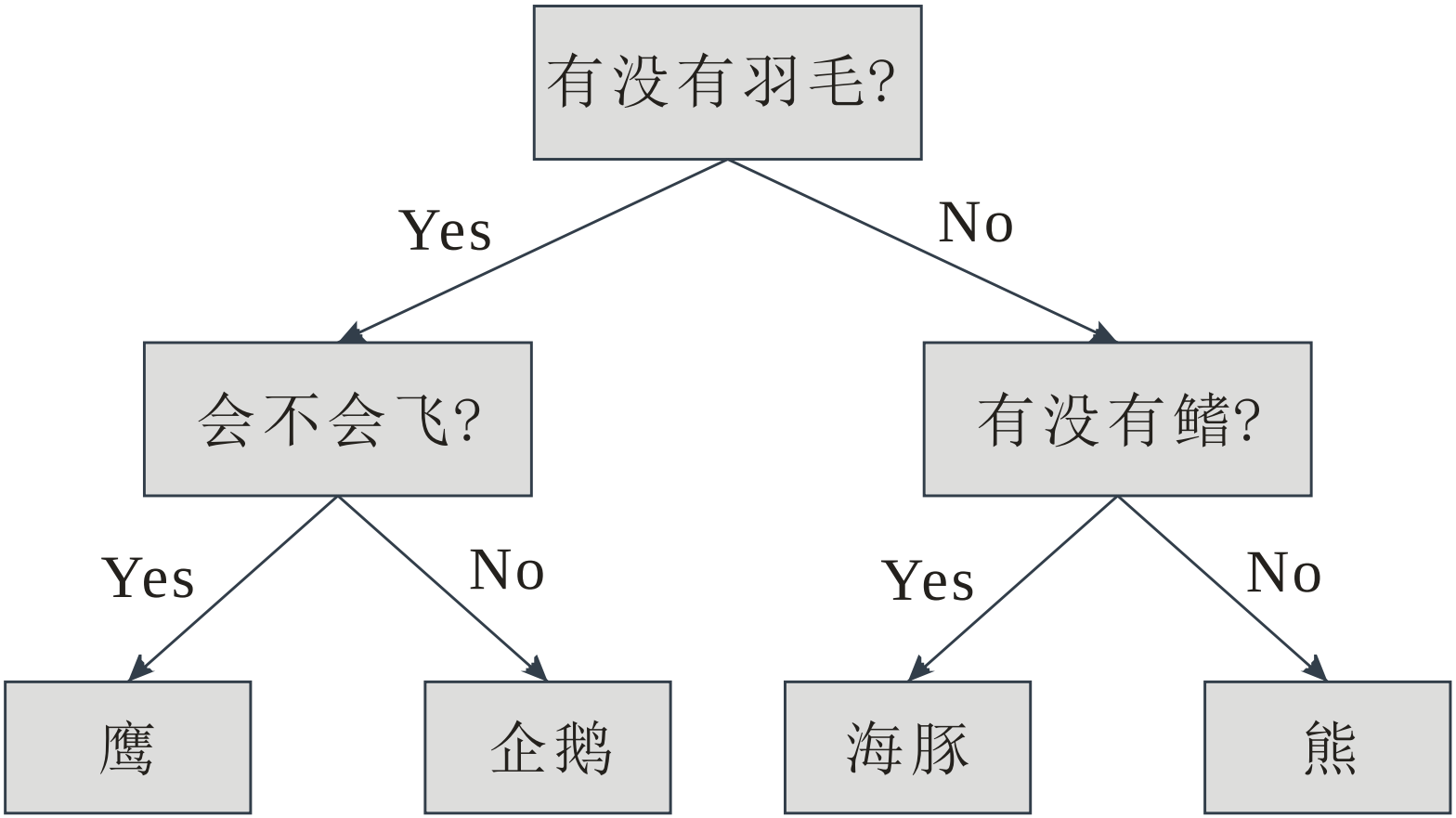

图6-1 区分四种动物的决策树

决策过程的最终结论对应判定的结果，如“熊”或“鹰”；决策过程中提出的每个判定问题都是对某个特征的测试，例如“有没有羽毛”、“会不会飞”；每个测试的结果或是导出最终结论，或是导出进一步的判定问题，其考虑范围是在上次决策结果的限定范围之内，例如在“有羽毛”之后再判断“会不会飞”，只是再考虑“有羽毛”的动物即鹰和企鹅会不会飞。

一般来说，一棵决策树包含一个根结点、若干个内部结点和若干个叶结点。叶结点对应于决策结果，其他每个结点对应于一个特征测试，每个结点包含的样本集合根据特征测试的结果被划分到子结点中，根结点包含整个训练样本集。从根结点到每个叶结点的路径对应一个判定测试序列。

决策树学习的目的是为了产生一棵泛化能力强的决策树。

决策树的构建可通过一个递归过程来完成：

---

generateTree(D, A):

输入：训练集 $D = \{(x_0, y_0), (x_1, y_1), ..., (x_{n-1}, y_{n-1})\}$, 特征集 $A=\{a_0, a_1, ..., a_{m-1}\}$

生成结点node;

if D中样本全属于同一类别，比如C，then

　将node标记为C类叶结点，return

if A为空集或D中样本在A上的取值都相同，then

　将node标记为叶结点，其类别标记为D中数目最多的类别，return

从A中选择最优划分特征，a_{opt}

for a_{opt}的每一个取值a^v_{opt}:

　为node生成一个分支，令D^v表示D中在a_{opt}特征上取值为a^v_{opt}的样本子集

　以generateTree(D^v, A - {a_{opt}})为分支结点

输出：以node为根结点的一棵决策树

图6-2 决策树构建流程图

在上面的算法中，有两种情况会导致递归返回：

(1) 当前结点包含的样本全属于同一类别，无需继续划分；

(2) 当前特征集为空，或是所有样本在所有特征上取值相同，无法划分。

在构建决策树的算法中，最关键的是如何选择最优划分特征，就如上面四种动物的例子，我们利用了三个特征进行分类:

表6-1 四种动物的特征

编号	有没有羽毛	会不会飞	有没有鳍	动物
0	否	否	否	熊
1	是	是	否	鹰
2	是	否	否	企鹅
3	否	否	是	海豚

在图6-1的决策树中，优先选择'有没有羽毛'这个特征进行划分，只用三层的决策树也就是两个问题就完成分类，如果选择另外两个特征的话，都需要多一个问题才可以。
由此可以看出，特征选择的合适，得出最终结果所需要的问题就少，相应的决策树层次少，算法效率就更高。那么如何选择最优划分特征呢？

2. 最优划分特征的选择

在生成决策树过程中，随着划分进行，我们希望决策树的分支结点所包含的样本尽可能属于同一类别，即结点的"纯度"越来越高，从而将无序的数据变得更加有序。不同的算法选择最优划分特征的依据不同：ID3算法依据信息增益，C4.5算法依据信息增益率，CART算法依据基尼指数。

2.1 信息增益

香农熵或简称为熵(entropy)是度量样本集合混乱程度最常用的指标，这个名字来源于信息论之父Claude Shannon。

假定当前样本集合*D*中有*K*个类别，其中每个类别*k*包含的样本数占总样本数的分率为p_k (k = 0, 1, ..., K-1)，则*D*的熵定义为：

$$S(D) = -\sum_{k=0}^{K-1} p_k \log_2 p_k$$

比如表6-1中的样本集，一共有4个类别，*K*=4，每一个类别的分率都是1/4，p_k=1/4，因而：

$$S(D) = -\sum_{k=0}^{K-1} p_k \log_2 p_k = -\sum_{k=0}^{3} \frac{1}{4} \log_2 \frac{1}{4} = -4 \times \frac{1}{4} \times (-2) = 2$$

假定特征a有V个可能的取值$\{a^0, a^1, \ldots, a^{V-1}\}$，若使用$a$来对样本集$D$进行划分，则会产生$V$个分支结点，其中第$v$个分支结点包含了$D$中所有在$a$特征上取值为$a^v$的样本，记为子集$D^v$。我们可根据熵的定义计算出$D^v$的熵，再考虑到不同分支结点所包含的样本数，给分支结点赋予权重$|D^v|/|D|$（这里$|D|$表示D中样本数目），于是可计算出特征a对样本集D进行划分所获得的“信息增益”（information gain）：

$$\mathrm{Gain}(D,a) = S(D) - \sum_{v=0}^{V-1} \frac{|D^v|}{|D|} S(D^v)$$

例如对表6-1中的样本集，如果用a=“有没有羽毛”这一特征进行划分，a有两个取值，V=2，假定a^0=“是”，a^1=“否”，则D^0包含鹰和企鹅，D^1包含熊和海豚。

D^0和D^1两个子集中都是含有两个不同类别的样本，因而$|D^0|=|D^1|=2$，各自的熵：

$$S(D^0)=S(D^1)=-\sum_{k=0}^{1}p_k\log_2 p_k=-\sum_{k=0}^{1}\frac{1}{2}\log_2\frac{1}{2}=-2\times\frac{1}{2}\times(-1)=1$$

则信息增益为：

$$\text{Gain}(D,a)=S(D)-\sum_{v=0}^{V-1}\frac{|D^v|}{|D|}S(D^v)$$

$$=2-\sum_{v=0}^{1}\frac{|D^v|}{|D|}S(D^v)=2-2\times\frac{2}{4}\times 1=1$$

如果用a=“会不会飞”这一特征划分，则D^0只包含鹰，D^1包含熊、企鹅和海豚，$|D^0|=|D^1|=2$。

$$S(D^0) = -\sum_{k=0}^{0} p_k \log_2 p_k = -1 \times \log_2 1 = 0$$

$$S(D^1) = -\sum_{k=0}^{2} p_k \log_2 p_k = -3 \times \frac{1}{3} \times \log_2 \frac{1}{3} = 1.585$$

信息增益为：

$$\text{Gain}(D, a) = S(D) - \sum_{v=0}^{1} \frac{|D^v|}{|D|} S(D^v) = 2 - \frac{1}{4} \times 0 - \frac{3}{4} \times 1.585 = 0.811$$

如果用*a*=“有没有鳍”这一特征划分信息增益也为0.811。信息增益越大，则意味着利用该特征进行划分所获得的“纯度提升”越大，划分后混乱程度越低。

用信息增益来进行最优划分特征选择，最优划分特征为：

$$a_{opt} = \underset{a \in A}{\operatorname{argmax}} \operatorname{Gain}(D, a)$$

按照这样的策略，对四个动物分类而言，应该首先选择“有没有羽毛”这一特征进行划分。著名的ID3决策树算法就是以信息增益为准则来选择划分特征的。

2.2 增益率

对表6-1样本集，如果把编号也作为一个特征的话，该特征的信息增益将高达2.0，远大于其他特征，因为编号作为特征将产生4个分支，每个分支结点仅包含一个样本，这些分支结点的纯度已达最大，熵为0。然而，这样的决策树没有任何实际价值，不具备泛化能力。

上面的例子以及ID3算法的使用经验表明：

利用信息增益选择特征对取值数目多的特征有所偏好。

为减少这种偏好可能带来的不利影响，C4.5算法不直接使用信息增益，而是使用“增益率”（gain ratio）来选择最优划分特征。

增益率定义为：

$$\text{GainRatio}(D,a)=\frac{\text{Gain}(D,a)}{\text{IV}(a)}$$

其中：

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

IV(a)称为特征a的固有值（intrinsic value）。特征a的可能取值数目越多（V越大），则IV(a)的值会越大。

但是，使用增益率作为特征选择的依据会对取值数目较少的特征有所偏好，因此在C4.5算法并不是直接选择增益率最大的特征，而是使用一个启发式策略：先从候选特征中找出信息增益高于平均水平的特征，再从中选择增益率最高的。

2.3 基尼指数

基尼值（Gini）亦称基尼不纯度（Gini impurity），也是描述混乱程度的一个指标，样本集*D*的基尼值定义为：

$$\mathrm{Gini}(D) = \sum_{k=0}^{K-1} p_k(1-p_k) = 1 - \sum_{k=0}^{K-1} p_k^2$$

直观上，Gini(D)反映了从样本集D中随机抽取两个样本，其类别不一样的概率，因此Gini(D)越小样本集D的纯度越高。

特征a的基尼指数（Gini index）定义为：

$$\text{GiniIndex}(D,a)=\sum_{v=1}^{V}\frac{|D^v|}{|D|}\text{Gini}(D^v)$$

著名的CART算法依据基尼指数选择最优划分特征，选择使划分后基尼指数最小的特征作为最优特征，即：

$$a_{opt}=\underset{a\in A}{\text{argmin}}\,\text{GiniIndex}(D,a)$$

3. 决策树算法的python实现

利用python实现决策树算法，以信息增益作为特征选择依据，即ID3算法，并对表6-1的样本集进行分类。

4. 剪枝处理

上面的算法在决策树生成过程中，只有还能细分就继续细分下去，这往往导致决策树节点过多，造成过似合。

剪枝（pruning）是决策树算法避免过拟合的主要手段，有预剪枝（prepruning）和后剪枝（post-pruning）。预剪枝可以有多种策略，比如对信息增益设置一个阀值，或设置一个叶子结点所包含的最小样本数目等。后剪枝先生成一棵完整的决策树，然后自底向上地对非叶结点进行考察，如果一个内部结点的划分所带来的信息增益小于指定阀值，则将其分支合并，或评估将该结点对应的子树替换为叶结点能带来决策树泛化性能提升，则将子树替换为叶结点等。

一般而言，预剪枝效率更高，但增大了欠拟合的风险；后剪枝的欠拟合风险很小，但训练过程的时间开销要大得多。

5. 利用scikit-learn实现决策树算法

在scikit-learn的tree模块中实现了DecisionTreeClassifier类用于决策树分类、DecisionTreeRegressor类用于决策树回归，另外，还提供了export_graphviz类用于决策树的可视化，下面以DecisionTreeClassifier为例介绍其用法:

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)

主要方法：

- decision_path(X): 返回测试样本X在决策树中的决策路径。
- fit(X, y, sample_weight = None): 由训练集(X, y)生成决策树。sample_weight指定样本权重，默认为None，即所有样本权重相同。
- predict(X): 预测X的类别。
- predict_proba(X): 预测X属于各种类别的概率。
- score(X, y, sample_weight = None): 返回对测试集(X, y)的预测正确率。

下面利用scikit-learn的DecisionTreeClassifier类对系统自带的Iris鸢尾花数据集构建决策树。

程序输出为：

```
X shape: (150, 4), y shape: (150,)
Accuracy on the training set is: 1.0
Accuracy on the testing set is: 0.9666666666666667
```

决策树对训练集的准确率100%，对测试集的准确率为96.7%。

为了决策树的可视化，在scikit-learn的tree模块提供了export_graphviz类，用于将决策树以DOT格式输出：

```
export_graphviz(decision_tree, out_file=None,
max_depth=None, feature_names=None, class_names=None,
label='all', filled=False, leaves_parallel=False, impurity=True,
node_ids=False, proportion=False, rotate=False, rounded=False,
special_characters=False, precision=3)
```

对于上面已经建立的iris决策树，可以使用如下代码输出为GraphViz dot格式：

```
tree.export_graphviz(dt, out_file = 'iris.dot',
                     feature_names = iris.feature_names,
                     class_names = iris.target_names,
                     filled = True)
```

这时在当前文件夹生成一个文件'iris.dot'，可以利用graphviz软件打开查看或将其转化为图片格式。

Graphviz软件的下载安装：

1. 进入Graphviz网站下载http://www.graphviz.org/download/，选择合适的版本，比如Windows下的Stable 2.38 Windows install packages，下载graphviz-2.38.msi文件，双击安装。

2. 环境变量的配置，在环境变量中添加graphviz安装目录中的bin目录，如D:\Program Files (x86)\Graphviz2.38\bin。

在windows命令行界面输入：dot -version，然后回车，如果显示图6.3所示的graphviz相关版本信息，则安装配置成功。

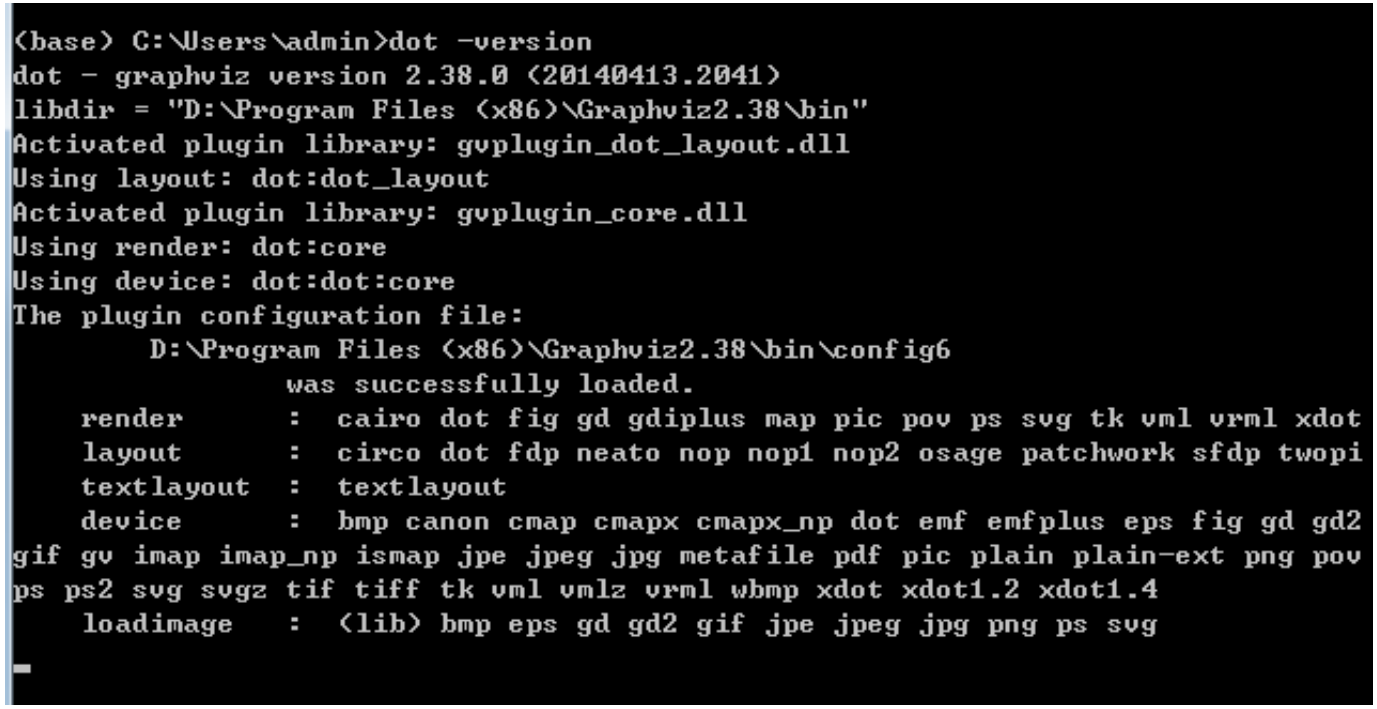

图6-3 Graphviz版本显示

Graphviz成功安装后，利用Graphviz软件可以直接打开刚才生成的iris.dot'文件，也可以在命令行执行如下命令生成png格式图片文件：

```
dot -Tpng iris.dot -o iris.png
```

或如下命令生成PostScript格式图片：

```
dot -Tps iris.dot -o iris.ps
```

或如下命令生成pdf格式文件：

```
dot -Tpdf iris.dot -o iris.pdf
```

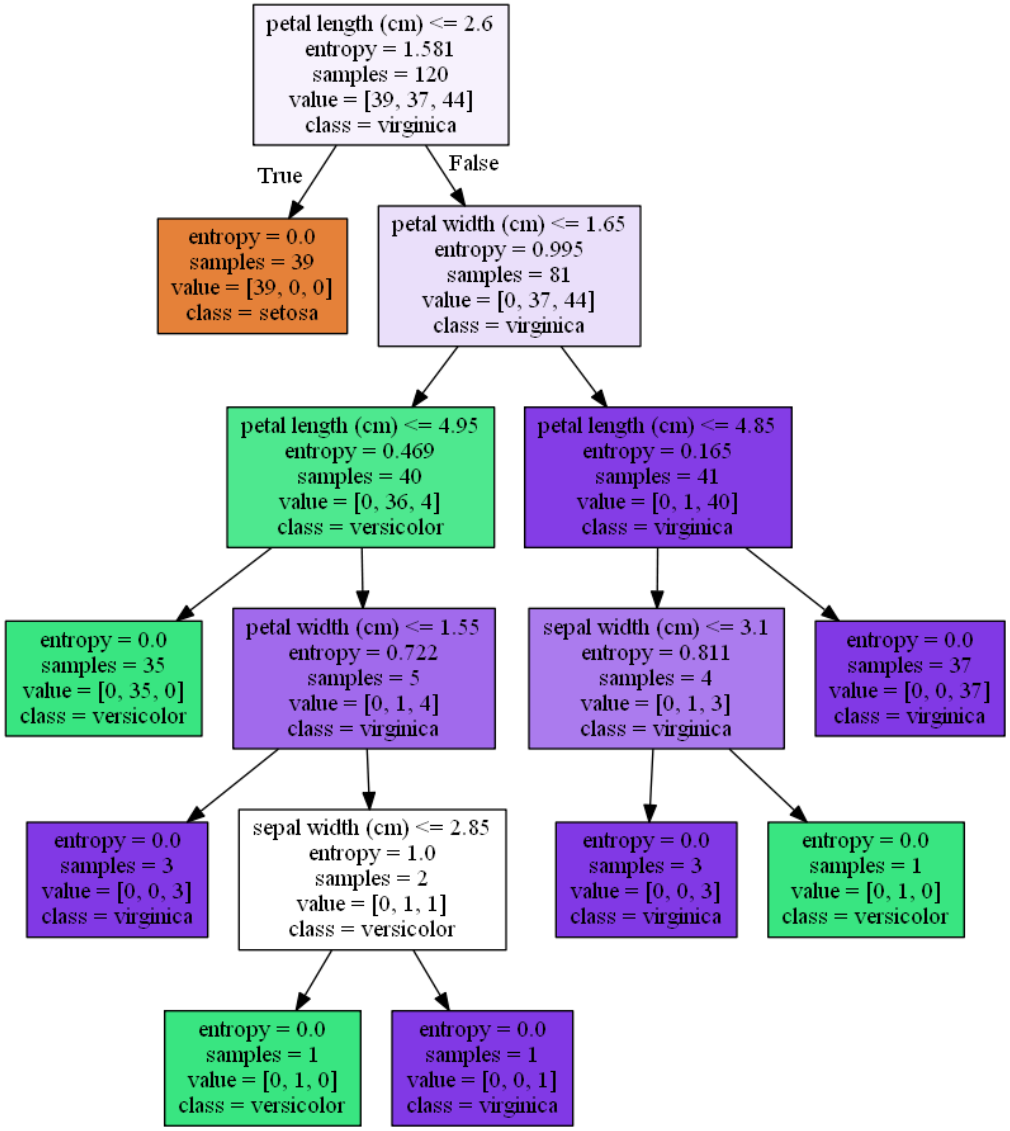

图6-4 利用120个样本的训练集生成的iris决策树