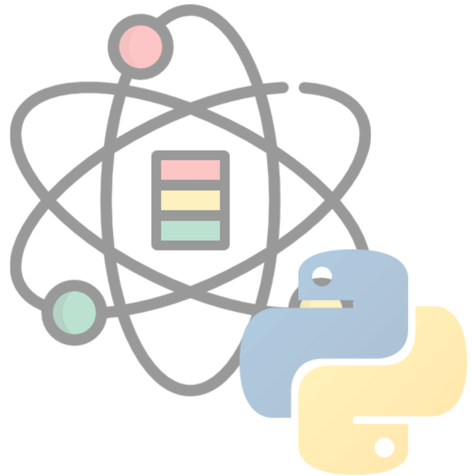
分类算法 (上)

Classification Algorithms - Part 1

范叶亮 Leo Van



目录

逻辑回归

逻辑回归

逻辑回归（Logistic Regression）的起源主要分为几个阶段，从开始想到 logistic 这个词，到发现 logistic function，再推导出 logit function，最后才命名 logistic regression。这些过程都是大量的研究者们共同努力发现的，只是在历史的长河中，很多人被渐渐遗忘了。

逻辑方程

广义逻辑曲线：

$$f(t) = A + \frac{K - A}{\left(C + Qe^{-Bt}\right)^{1/v}} \tag{1}$$

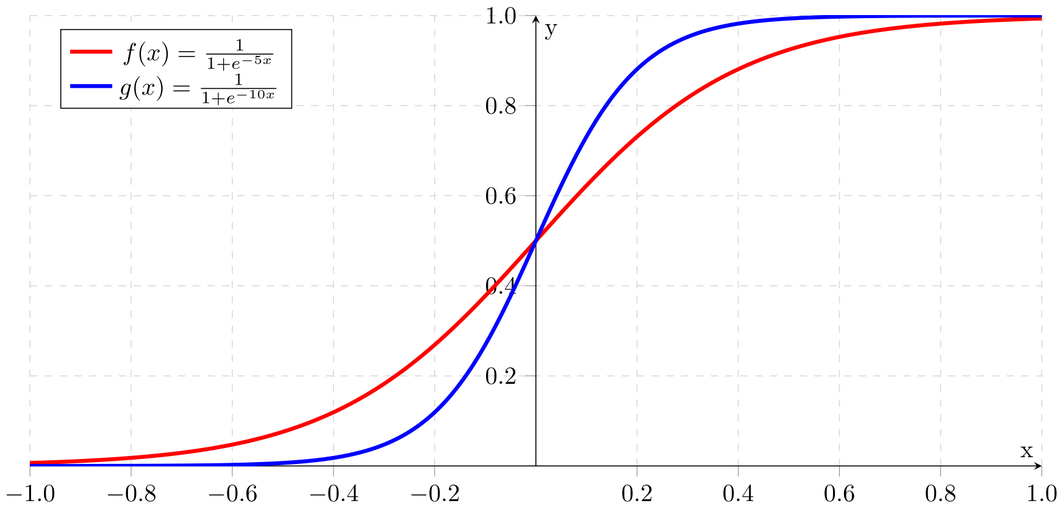
其中：

A 为曲线下界
K 为曲线上界
B 为增长率
$v > 0$ 影响曲线的增长率
Q 与 $f(0)$ 相关
C 一般取值为 1

Sigmoid曲线：

$$f(t) = \frac{1}{1 + e^{-t}} = \frac{e^t}{1 + e^t} \tag{2}$$

$$f'(t) = \frac{e^t}{\left(1 + e^t\right)^2} = f(t)\left(1 - f(t)\right) \tag{3}$$



逻辑回归模型

逻辑回归（Logistic Regression）模型是一种分类模型，这里，我们利用条件概率分布 $P(Y|X)$ 表示，这里随机变量 X 取值为实数，随机变量 Y 取值为 1 或 0。我们通过监督学习的方法来估计模型参数：

$$P(Y=1|x) = \frac{\exp(w \cdot x + b)}{1 + \exp(w \cdot x + b)} \tag{4}$$

$$P(Y=0|x) = \frac{1}{1 + \exp(w \cdot x + b)} \tag{5}$$

其中：$x \in \mathbb{R}^n$ 为输入，$Y \in 0, 1$ 为输出，$w \in \mathbb{R}^n$ 和 $b \in \mathbb{R}$ 为模型参数，w 称之为权重，b 称之为偏置。

逻辑回归模型

现在考虑一下逻辑回归模型的特点，一个事件发生的几率（odds）是指这个事件发生的概率与事件不发生的概率的比值。如果事件发生的概率为 p，那么这个事件发生的几率为 $\frac{p}{1-p}$，此时我们定义这个事件的对数几率（log odds）或 logit 函数为：

$$\operatorname{logit}(p) = \log \frac{p}{1-p} \tag{6}$$

因此，对于逻辑回归模型而言，其对数几率为：

$$\log \frac{P(Y=1|x)}{1-P(Y=1|x)} = w \cdot x \tag{7}$$

所以，在逻辑回归模型中，输出 $Y=1$ 的对数几率是输入 x 的线性函数。因此，如果线性函数的值越接近正无穷，则 $P(Y=1|x)$ 的概率值就越接近 1，反之，如果线性函数的值越接近负无穷，则 $P(Y=1|x)$ 的概率值就越接近 0。

逻辑回归

在 Python 中，我们利用 scikit-learn 包中的 `LogisticRegression` 构建逻辑回归模型，函数定义如下：

```
sklearn.linear_model.LogisticRegression(
    penalty='deprecated', *, C=1.0, l1_ratio=0.0, dual=False, tol=0.0001, fit_intercept=True, intercept_scaling=1,
    class_weight=None, random_state=None, solver='lbfgs', max_iter=100, verbose=0, warm_start=False, n_jobs=None)
```

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
https://scikit-learn.org/stable/modules/tmp/sklearn.linear_model.LogisticRegression.html

逻辑回归

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

X, y = load_iris(return_X_y=True)
clf = LogisticRegression(random_state=0).fit(X, y)
clf.predict(X[:2, :])
```

```
array([0, 0])
```

```
clf.predict_proba(X[:2, :])
```

```
array([[9.81778522e-01, 1.82214641e-02, 1.43182484e-08],
       [9.71708980e-01, 2.82909905e-02, 2.98868576e-08]])
```

```
clf.score(X, y)
```

```
0.9733333333333334
```

决策树

决策树

决策树（Decision Tree）是机器学习和数据挖掘中的一套分类和回归方法。决策树是由节点和有向边构成的树形分类模型，其中树的叶子节点表示某个分类，非叶子结点表示一个用于树枝分叉的特征属性。

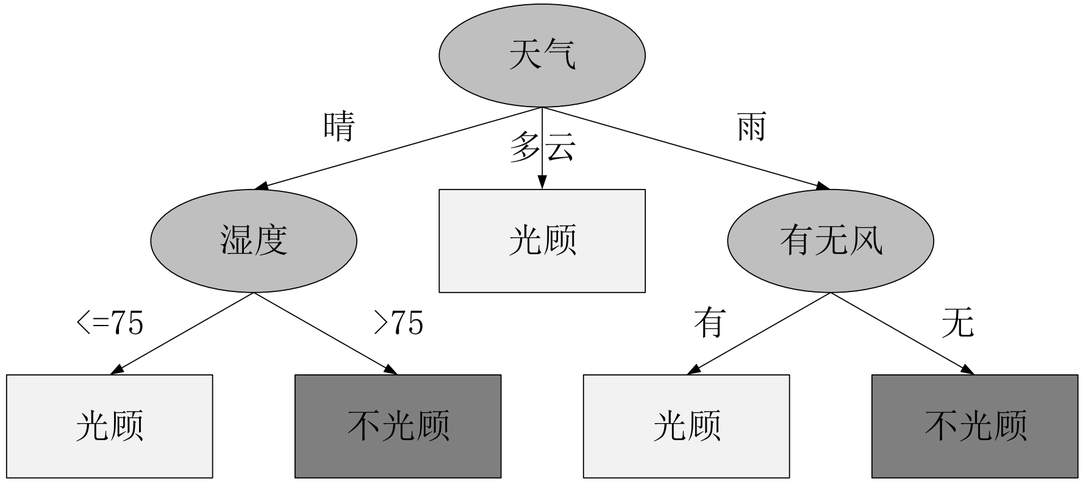
例如：高尔夫俱乐部的经理通过周天气预报寻找什么时候人们会打高尔夫，以适时调整雇员数量。在 2 周时间内我们得到以下记录：天气状况有晴，云和雨；气温用华氏温度表示；相对湿度用百分比；还有有无风；以及顾客是不是在这些日子光顾俱乐部，如右表所示。

日期	天气	气温	湿度	风	光顾
1	晴	85	85	无	否
2	晴	80	90	有	否
3	多云	83	78	无	是
4	雨	70	96	无	是
5	雨	68	80	无	是
6	雨	65	70	有	否
7	多云	64	65	有	是
…	…	…	…	…	…

决策树

Golf 数据集对应得到的决策树如图所示。根节点代表整个训练样本集，通过在每个节点对某个属性的测试验证，递归得将数据集分成更小的数据集。某一节点对应的子树对应着原数据集中满足某一属性的部分数据集。递归过程一直进行下去，直到某一节点对应的子树对应的数据集都属于同一个类为止。决策树的生成主要有三个步骤：

- 特征的选择：特征选择是指从数据集中的多个属性特种中选择具有分类能力的特征。不同的特征选择策略将导致不同决策树的生成。
- 决策树生成：决策树生成是指利用选择的特征，递归的构建决策树。
- 决策树剪枝：决策树剪枝是指为了防止过拟合现象，对于过于复杂的决策树进行简化的过程。



决策树特征选择

特征选择主要解决的问题是用来进行划分特征空间的特征的顺序以及特征自身的划分依据等。根据数据集可知，是否有客户光顾高尔夫俱乐部受到天气，气温，湿度和有无风四个变量的制约。其中，天气和有无风为离散型变量，气温和湿度为连续型变量。那么在构建决策树时，会有两个基本问题：

1. 依次选择哪些特征作为划分节点，才能够保证每个节点都能够具有最好的分类能力？
2. 对于连续型变量，选择什么值作为划分依据？

决策树主要从信息论的角度处理这两个问题，具体的选择依据有**信息增益**，**信息增益率**和 **Gini 系数**等。

熵 Entropy

首先我们需要理解什么是熵（Entropy）。熵最早是用来表示物理学中一个热力系统无序的程度，后来依据香农的信息论，熵用来衡量一个随机变量的不确定性程度。对于一个随机变量 X，其概率分布为：

$$P\left(X=x_i\right)=p_i,\quad i=1,2,\ldots,n \tag{8}$$

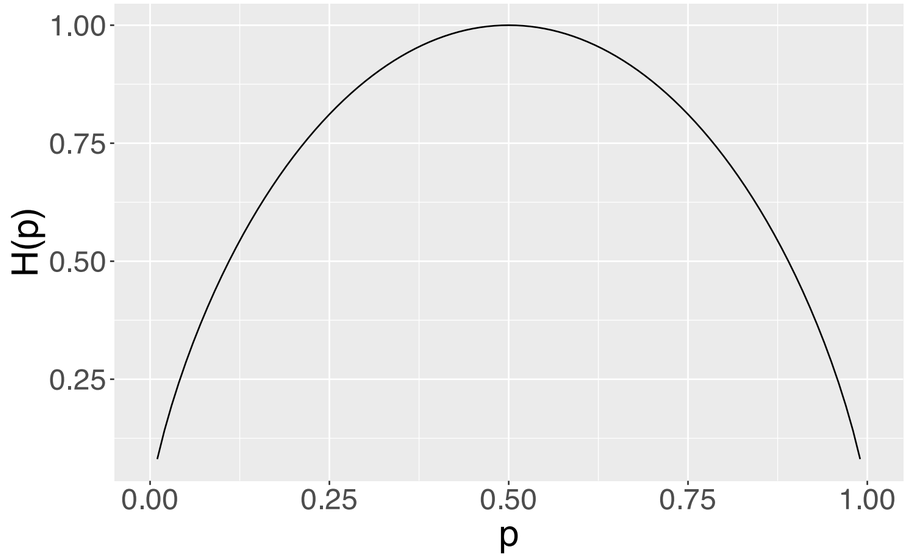
则随机变量 X 的熵定义如下：

$$H\left(X\right)=-\sum_{i=1}^{n}p_i\log p_i \tag{9}$$

例如抛一枚硬币，假设硬币正面向上 $X=1$ 的概率为 p，硬币反面向上 $X=0$ 的概率为 $1-p$。则对于抛一枚硬币那个面朝上这个随机变量 X 的熵为：

$$H\left(p\right)=-p\log p-\left(1-p\right)\log\left(1-p\right) \tag{10}$$

则熵 $H\left(p\right)$ 随概率 p 变化如图所示：



信息增益

对于随机变量 (X, Y)，其联合概率分布为：

$$P\left(X=x_i, Y=y_i\right)=p_{ij}, \quad i=1,2,\ldots,n; \quad j=1,2,\ldots,m \tag{11}$$

则条件熵 $H\left(Y|X\right)$ 表示在已知 X 的条件下 Y 的不确定性，定义如下：

$$H\left(Y|X\right)=\sum_{i=1}^{n} p_i H\left(Y|X=x_i\right) \tag{12}$$

信息增益（information gain）表示在已知特征 X 对于类 Y 的不确定性的减少程度。则特征 A 对于数据集 D 的信息增益 $G\left(D, A\right)$，可以用数据集的熵 $H\left(D\right)$ 和已知特征 A 的前提下的经验熵 $H\left(D, A\right)$ 之差来表示：

$$G\left(D, A\right)=H\left(D\right)-H\left(D, A\right) \tag{13}$$

信息增益

利用高尔夫俱乐部光顾数据，计算变量“天气”对于系统的信息增益。首先计算整个系统的熵 $H(D)$，

$$H(D) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940 \tag{14}$$

变量“天气”共有 3 种取值类型：“晴”，“雨”和“多云”，因此在已知变量“天气”的前提下，系统的条件熵 $H(D, A)$，

$$\begin{aligned} H(D, A) =& \frac{5}{14}H(D_1) + \frac{5}{14}H(D_2) + \frac{4}{14}H(D_3) \\ =& \frac{5}{14}\left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) + \frac{5}{14}\left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right) \\ &+ \frac{4}{14}\left(-\frac{4}{4}\log_2\frac{4}{4} - \frac{0}{4}\log_2\frac{0}{4}\right) \\ =& 0.347 + 0.347 + 0 = 0.693 \end{aligned} \tag{15}$$

其中 D_1，D_2 和 D_3 分别表示当“天气”取值“晴”，“雨”和“多云”时的子系统。则在已知变量“天气”的前提下，系统的熵增益为 $G(D, A) = 0.940 - 0.693 = 0.247$。

信息增益率

信息增益率（information gain ratio）为特征 A 对于数据集 D 的信息增益 $G\left(D,A\right)$ 与数据集 D 关于特征 A 的熵 $H_A\left(D\right)$ 的比：

$$G_r\left(D,A\right)=\frac{G\left(D,A\right)}{H_A\left(D\right)} \tag{16}$$

其中，$H_A\left(D\right)=-\sum_{i=1}^{n}\frac{\left|D_i\right|}{\left|D\right|}\log_2\frac{\left|D_i\right|}{\left|D\right|}$，$D_i$ 表示特征 A 的第 i 种取值。

则高尔夫俱乐部光顾数据集关于特征“天气”的熵 $H_A\left(D\right)$：

$$H_A\left(D\right)=-\frac{5}{14}\log_2\frac{5}{14}-\frac{5}{14}\log_2\frac{5}{14}-\frac{4}{14}\log_2\frac{4}{14}=1.577 \tag{17}$$

则信息增益率 $G_r\left(D,A\right)=\frac{0.247}{1.577}=0.156$。

Gini 指数

对于一个随机变量 X，其概率分布为：

$$P\left(X=x_i\right)=p_i,\quad i=1,2,\ldots,n \tag{18}$$

则随机变量 X 的 Gini 指数定义如下：

$$\operatorname{Gini}\left(p\right)=\sum_{i=1}^{n}p_i\left(1-p_i\right)=1-\sum_{i=1}^{n}p_i^2 \tag{19}$$

假设特征 A 有 n 中取值，可以将数据集划分为 $D_1,D_2,\ldots,D_n$，则在已知特征 A 的前提下，数据集 D 的 Gini 指数定义为：

$$\operatorname{Gini}\left(D,A\right)=\sum_{i=1}^{n}\frac{\left|D_i\right|}{\left|D\right|}\operatorname{Gini}\left(D_i\right) \tag{20}$$

则特征“天气”的 Gini 指数：

$$\begin{aligned}
&\operatorname{Gini}\left(D,A\right)\\
=&\frac{5}{14}\operatorname{Gini}\left(D_1\right)+\frac{5}{14}\operatorname{Gini}\left(D_2\right)+\frac{4}{14}\operatorname{Gini}\left(D_3\right)\\
=&\frac{5}{14}\left[1-\left(\frac{2}{5}\right)^2-\left(\frac{3}{5}\right)^2\right]+\\
&\frac{5}{14}\left[1-\left(\frac{3}{5}\right)^2-\left(\frac{2}{5}\right)^2\right]+\\
&\frac{4}{14}\left[1-\left(\frac{4}{4}\right)^2-\left(\frac{0}{4}\right)^2\right]\\
=&0.171+0.171+0\\
=&0.343
\end{aligned} \tag{21}$$

其中 D_1，D_2 和 D_3 分别表示当“天气”取值“晴”，“雨”和“多云”时的子系统。

连续型变量处理方法

上面介绍了对于离散型变量如何计算熵增益，熵增益率和 Gini 指数，但对于连续型变量如何计算着三个指标呢?

把需要处理的样本（对应根节点）或样本子集（对应子树）按照连续变量的大小从小到大进行排序，假设该属性对应的不同的属性值一共有 N 个，那么总共有 $N-1$ 个可能的候选分割阈值点，每个候选的分割阈值点的值为上述排序后的属性值链表中两两前后连续元素的中点，那么我们的任务就是从这个 $N-1$ 个候选分割阈值点中选出一个，使得前面提到的信息论标准最大。

例如，对于高尔夫数据集，我们来处理温度属性，来选择合适的阈值。首先按照温度大小对对应样本进行排序如表：

温度	**64**	**65**	**68**	**69**	**70**	**71**	**72**	**72**	**75**	**75**	**80**	**81**	**83**	**85**
光顾	是	否	是	是	是	否	否	是	是	是	否	是	是	否

连续型变量处理方法

那么可以看到有 13 个可能的候选阈值点，比如 $middle[64, 65]$，$middle[65, 68]$，…，$middle[83, 85]$。例如，选择 $middle[71, 72]$ 时，当温度 < 71.5 是，有 4 个“是”和 2 个“否”，当温度 > 71.5 时，有 5 个“是”和 3 个“否”，此时的条件熵 $H(D, A)$，

$$
\begin{aligned}
H(D, A) =& \frac{6}{14} H(D_1) + \frac{8}{14} H(D_2) \\
=& \frac{6}{14}\left(-\frac{2}{6}\log_2\frac{2}{6} - \frac{4}{6}\log_2\frac{4}{6}\right) \\
&+ \frac{8}{14}\left(-\frac{3}{8}\log_2\frac{3}{8} - \frac{5}{8}\log_2\frac{5}{8}\right) \\
=& 0.393 + 0.545 \\
=& 0.939
\end{aligned}
\tag{22}
$$

因此，通过对所有可能的阈值点计算条件熵可以找出信息增益最大阈值点。连续变量的熵增益率和 Gini 指数计算类似。

决策树生成 - CART 算法

Algorithm 1 CART 算法

Require: 数据集 D，特征集 A，阈值 ϵ
Ensure: 决策树 $Tree$

procedure CART(D, A, ϵ)
  $Tree \leftarrow \{\}$
  if All data in D belong to class C_k **then**
    $Tree \leftarrow$ single node with class of C_k
    terminate
  end if
  for all attribute $D_i \in D$ **do**
    Compute information gain if we split on D_i
  end for
  $G_{min} \leftarrow$ Minimum Gini
  //接下文
end procedure

procedure CART(D, A, ϵ)
  //接上文
  if $G_{min} > \epsilon$ **then**
    $C \leftarrow$ Class with most examples
    $Tree \leftarrow$ single node with C
    terminate
  end if
  $A_{best} \leftarrow$ Best attribute with G_{min}
  $Tree \leftarrow$ Create a decision node that in the root with A_{best}
  $D_s \leftarrow$ All sub datasets of D based on attribute A_{best}
  for all data $D_{si} \in D_s$ **do**
    $Tree_{si} \leftarrow$ CART(D_{si}, A_{si}, ϵ)
    Attach $Tree_{si}$ to the $Tree$
  end for
  return $Tree$
end procedure

决策树剪枝

决策树对于训练样本来说，可以得到一个 100% 准确的分类器。算法生成的决策树非常的详细而且庞大，每个属性都被详细地加以考虑。但是如果训练样本中包含了一些错误，按照前面的算法，这些错误也会 100% 被决策树学习了，这就产生了过拟合现象。因此，为了解决这个问题，我们需要对生成的决策树进行简化，这个简化的过程就称之为剪枝。

决策树剪枝

决策树的剪枝有多种算法，但其目的都是为了平衡决策树的准确性和复杂度。假设一棵树 T 的叶子节点的个数为 $|T|$，t 为树 T 的叶子节点，这个叶子节点共有 N_t 个样本点，其中数据类 c 的样本点共有 N_{tc} 个，则可以定义决策树学习的损失函数为：

$$Loss\left(T,\alpha\right)=\sum_{t=1}^{|T|}N_tH_t\left(T\right)+\alpha|T| \tag{23}$$

其中，$\alpha\geq 0$ 为参数，$H_t\left(T\right)$ 为叶子节点 t 上的经验熵。上式中的前项记做：

$$E\left(T\right)=\sum_{t=1}^{|T|}N_tH_t\left(T\right) \tag{24}$$

$$Loss\left(T,\alpha\right)=E\left(T\right)+\alpha|T| \tag{25}$$

其中 $E\left(T\right)$ 表示模型在训练集上的预测误差，$\alpha|T|$ 表示模型的复杂度。当 α 较大时，意味着我们希望选择复杂度较小的决策树（叶子节点较少），反之，意味着我们希望选择对于训练集的预测误差较小但相对复杂度较大的决策树。当 α 的值确定后，我们可以通过最小化损失函数来得到最终的决策树。

决策树

在 Python 中，我们利用 scikit-learn 包中的 `DecisionTreeClassifier` 构建决策树模型，函数定义如下：

```
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, class_weight=None, ccp_alpha=0.0, monotonic_cst=None)
```

https://scikit-learn.org/stable/modules/tree.html
https://scikit-learn.org/stable/modules/tmp/sklearn.tree.DecisionTreeClassifier.html

决策树

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

X, y = load_iris(return_X_y=True)
clf = DecisionTreeClassifier(random_state=0)
clf = clf.fit(X, y)
clf.predict(X[:2, :])
```

```
array([0, 0])
```

```
clf.predict_proba(X[:2, :])
```

```
array([[1., 0., 0.],
       [1., 0., 0.]])
```

```
clf.score(X, y)
```

```
1.0
```

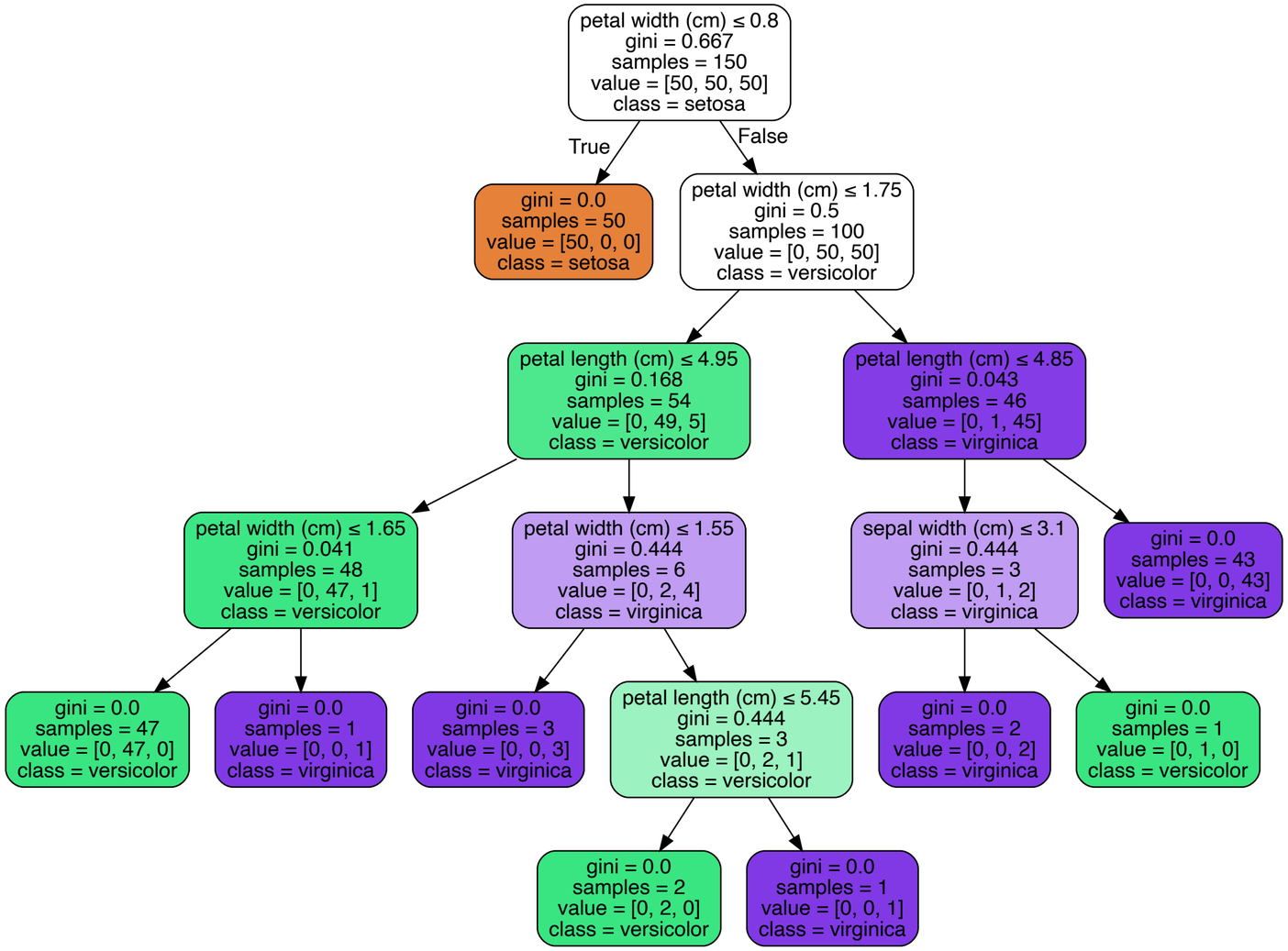
在训练过后，可以利用 `plot_tree` 函数绘制决策树：

```
from sklearn.tree import plot_tree
iris = load_iris()
plot_tree(clf.fit(iris.data, iris.target))
```

利用 Graphviz 的 `export_graphviz` 绘制美观的决策树：

```
import graphviz
from sklearn.tree import export_graphviz
iris = load_iris()
dot_data = export_graphviz(
    clf, out_file=None, filled=True,
    feature_names=iris.feature_names,
    class_names=iris.target_names,
    special_characters=True, rounded=True)
graph = graphviz.Source(dot_data, format='svg')
graph.render('iris-decision-tree')
```

决策树


petal width (cm) ≤ 0.8
gini = 0.667
samples = 150
value = [50, 50, 50]
class = setosa
True
False
gini = 0.0
samples = 50
value = [50, 0, 0]
class = setosa
petal width (cm) ≤ 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
class = versicolor
petal length (cm) ≤ 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor
petal length (cm) ≤ 4.85
gini = 0.043
samples = 46
value = [0, 1, 45]
class = virginica
petal width (cm) ≤ 1.65
gini = 0.041
samples = 48
value = [0, 47, 1]
class = versicolor
petal width (cm) ≤ 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
class = virginica
sepal width (cm) ≤ 3.1
gini = 0.444
samples = 3
value = [0, 1, 2]
class = virginica
gini = 0.0
samples = 43
value = [0, 0, 43]
class = virginica
gini = 0.0
samples = 47
value = [0, 47, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
petal length (cm) ≤ 5.45
gini = 0.444
samples = 3
value = [0, 2, 1]
class = versicolor
gini = 0.0
samples = 2
value = [0, 0, 2]
class = virginica
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
gini = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica

感谢倾听