机器学习讲义 (L18) 决策树

授课教师：王贝伦 / 助教：张嘉琦，黄旭，谈笑，徐浩卿

1 决策树

之前我们讨论的机器学习方法，大多是十分抽象的数学模型，例如线性模型、KNN 等，它们能给出结论，但依据通常非常简单（如用一条直线进行决策），给不出详细的决策理由。

决策树来源于日常生活中的决策方法，例如我们决定是否要出门打球时，会依次考虑天气、人数、场地等条件，最后得到出去还是不出去的决策结果。这种模型与我们人的思考方式很相似，因而训练完成后也能理解模型每一步分类的依据。一个简化的决策树模型如下图所示。

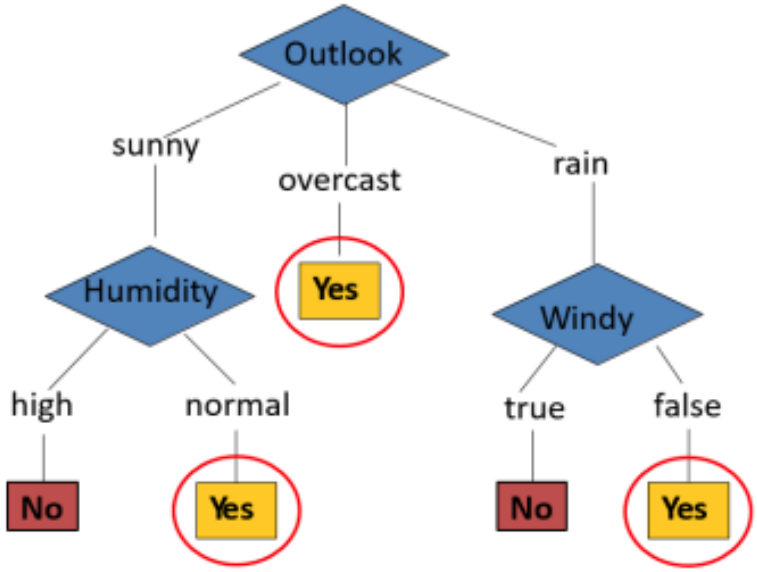

Figure 1: 决策树模型示意

1.1 构建决策树

在构建决策树的时候，其每个节点都是一个判断条件。假设我们将训练集送入决策树，上一步得到的集合经过判断后被分为两个子集。由于我们最终目的是要进行决策，即一个叶节点应该对应一个唯一的决策结果，那么我们自然希望每次判断后能尽可能将不同的决策结果分开。

如果集合中标签（正确的决策结果）有一种占了大多数，我们称这个集合“纯净度”高；反之则“纯净度”低。如何量化“纯净度”呢？信息论给了我们一些启发，在下一节我们将简单了解信息论的相关内容。

2 信息论简介

在详细介绍如何构建决策树模型前，我们先简单介绍一下信息论的相关内容。

2.1 信息

在对一个随机变量进行观察时，怎么知道一个观察值带给我们的信息有多少呢？香农提出自信息的概念：

定义：自信息

假设事件 $\omega \in \Omega$ 发生的概率是 $P(\omega)$，则自信息

$$I(\omega) = -\log_2 P(\omega) \tag{1}$$

直观地来看，事件发生的概率越低，被观测到时其自信息就越大。这比较符合我们的常识，因为如果一个事件发生的概率是 1，那它就能被完全预测，观测值也就没有任何意义；反之如果一个事件发生概率非常低，但是它发生了，那么这个观测蕴含的信息应该就较高。

例 2.1. 现有一枚正反面图案不同的均质硬币，抛掷一次，若记事件 A：朝上的为正面图案。那么自信息 $I(A) = -\log_2 P(A) = 1$。

现在假设这枚硬币正反面图案相同，则 $I'(A) = 0$。

2.2 熵

有了对单次观测信息量的定义后，我们可以顺势定义对这个随机变量不确定性的度量，借用物理学中熵的概念：

定义：信息熵

假设随机变量 $X \in \mathcal{X}$，则 X 的信息熵

$$H(X) = E[I(X)] \tag{2}$$

如果 X 是离散的，例如集合 $\mathcal{X} = \{1, 2, \cdots, k\}$，则

$$H(X) = -\sum_{i=1}^{k} P(X = i) \log_2 P(X = i) \leq \log_2 k \tag{3}$$

当且仅当 $P(X = i) = \dfrac{1}{k}$ 时取等号。这说明，当每个事件发生的概率相同时这个概率空间的熵最大，这是因为此时对下一个观测值完全无法预测。

例 2.2. 投掷一颗均值六面骰，记随机变量 X 为朝上的点数，则

$$H(X) = \log_2 6 \tag{4}$$

若给定 Y 会对 X 的概率产生影响，也可以定义条件信息熵：

定义：条件信息熵

对于随机变量 X,Y，给定 $Y = y_i$ 时 X 的信息熵为

$$H(X|Y = y_i) = E_{X|Y=y_i}[I(X|Y = y_i)] \tag{5}$$

其中 $I(X|Y = y_i) = -\log_2 P(X|Y = y_i)$。则给定 Y 时 X 的条件信息熵为

$$H(X|Y) = E_Y[H(X|Y = y_i))] \tag{6}$$

2.3 信息增益

给定变量 Y，如果它与 X 有一定联系，即它带来了一定的信息，则 $X|Y$ 的条件信息熵应该会减少。将信息熵减少的值定义为信息增益：

定义：信息增益

对于随机变量 X,Y，以 Y 为条件的信息增益

$$IG(X, Y) = H(X) - H(X|Y) \tag{7}$$

3 构建决策树

我们可以用信息熵来量化“纯净度”：熵越低，纯净度越高。

我们以特征均为布尔值的情况为例来讨论如何构建决策树。假设 $x_1, x_2 \in \{T, F\}$ 是样本的两个特征，$y \in \{+, -\}$ 是样本的类别。现在需要从 x_1, x_2 中选出一个特征，将为 T 的分为一个子集，为 F 的分为另一个，构成一个决策节点。我们希望选择的这个特征能给类别 y 最多信息量，即选择信息增益最大的条件 x_k：

$$\begin{aligned} x_k &= \underset{i}{\operatorname{argmax}}\ IG(y, x_i) \\ &= \underset{i}{\operatorname{argmax}}\ H(y) - [P(x_i = T)H(y|x_i = T) \\ &\quad + P(x_i = F)H(y|x_i = F)] \end{aligned} \tag{8}$$

选出 x_k 后，对两个子集继续按上述规则构建下一层决策节点，直至子集中类别只剩一种。

例 3.1. 这里我们给出一个构建决策节点的例子来帮助理解。假设训练集如下图所示，其中 x_1, x_2 为布尔型特征，y 为类别，*Count* 为样本点计数。

X1	X2	Y	Count
T	T	+	2
T	F	+	2
F	T	-	5
F	F	+	1

Figure 2: 训练数据集

我们要从两个特征中选出一个，按取值分为 *2* 个子集，构成一个决策节点。接下来我们以 x_1 为例计算其信息增益所需的信息熵：

$$\begin{aligned} H(y) &= -P(y=+)\log P(y=+) - P(y=-)\log P(y=-) \\ &= -\frac{5}{10}\log\frac{5}{10} - \frac{5}{10}\log\frac{5}{10} \\ &= 1 \end{aligned} \tag{9}$$

$$\begin{aligned} H(y|x_1=T) &= -P(y=+|x_1=T)\log P(y=+|x_1=T) \\ &\quad - P(y=-|x_1=T)\log P(y=-|x_1=T) \\ &= -1\times\log 1 - 0\times\log 0 \\ &= 0 \end{aligned} \tag{10}$$

$$H(y|x_1=F) = -\frac{5}{6}\log\frac{5}{6} - \frac{1}{6}\log\frac{1}{6} = 0.65$$

$$\begin{aligned} H(y|x_1) &= P(x_1=T)H(y|x_1=T) + P(x_1=F)H(y|x_1=F) \\ &= \frac{4}{10}\times 0 + \frac{6}{10}\times(-\frac{5}{6}\log\frac{5}{6} - \frac{1}{6}\log\frac{1}{6}) \\ &= 0.39 \end{aligned} \tag{11}$$

那么 x_1 带来的信息增益为

$$\begin{aligned} IG(y|x_1) &= H(y) - H(y|x_1) \\ &= 1 - 0.39 \\ &= 0.61 \end{aligned} \tag{12}$$

同样我们计算出 x_2 带来的信息增益

$$IG(y|x_2) = 0.12 \tag{13}$$

因为 $x_1 > x_2$，于是我们在这里选择 x_1，分为两子集：一类含有 *4* 个样本，其类别全为 $+$；另一类含有 *6* 个样本，其中除 *1* 个 $+$ 外均为 $-$。对这两子集继续按上述规则选择特征（在本例中只剩最后一种）直到子集只剩一种类别（本例中有一类已经达成这个条件）。

3.1 讨论

- 决策树是一个非线性分类/回归模型。
- 决策树使用简单：对任意的数据集使用决策树，都可以构建出一棵正确分类所有点的决策树，因为决策树的最坏情况就是枚举。同时，决策树的可解释性也较强，因为用户能够根据从根节点到叶节点的路径了解决策的依据。
- 决策树有一些弊端：修改一些训练集的样本点，整棵树可能会发生很大变化（方差高）；层级结构使得上方节点产生的错误会传递到下层节点，并不断传播。
- 为了防止决策树分支太多、方差太高，有一些可行的解决办法：
  - 当分出的子集不能提高决策效果时，应该停止分支。
  - 完全构建好一棵决策树（即使最终效果是枚举），之后删除一些节点（剪枝）。

4 集成学习

在机器学习中，集成学习集成了多个学习算法，以获得比其中任意一个单独的学习算法更好的预测效果。简单来说，集成学习就是将多个学习器通过某种策略结合成一个整体，如同一个黑盒，可以根据输入数据来预测结果。通过集成，可以达到减小方差、偏差或改进预测的效果。如果所有的单个学习器都是同类的，比如都为决策树，那么这样的集成就是同质（Homogeneous）的；如果，学习器不全是同类的，比如又有决策树又有神经网络，那么这样的集成就是异质的（Heterogeneous）。集成方法可以分为两类：序列集成方法和并行集成方法

- 序列集成方法中，参与训练的个体学习器按顺序生成。序列方法的原理是利用个体学习器之间的依赖关系，通过对之前训练中错误预测的样本给予较高的权重，可以提高整体的预测效果。比如 Boosting 算法。
- 并行集成方法中，参与训练的个体学习器可以并行生成。并行方法的原理是利用个体学习器之间的独立性，通过平均可以降低方差。比如即将介绍的聚合算法。

总体来说，集成学习的泛化能力是远大于单个学习器的泛化能力的。根据经验，当个体学习器之间有更多差异性时，集成往往会产生更好的效果。要获得好的集成，个体学习器需要具有一定的准确性，同时要有一定的多样性，即具有差异。如图3所示，h_i 表示第 i 个分类器，✓表示分类正确，$\times$ 表示分类错误。那么当个体学习器准确率较高且多样性较高时，集成的性能得到了提升。

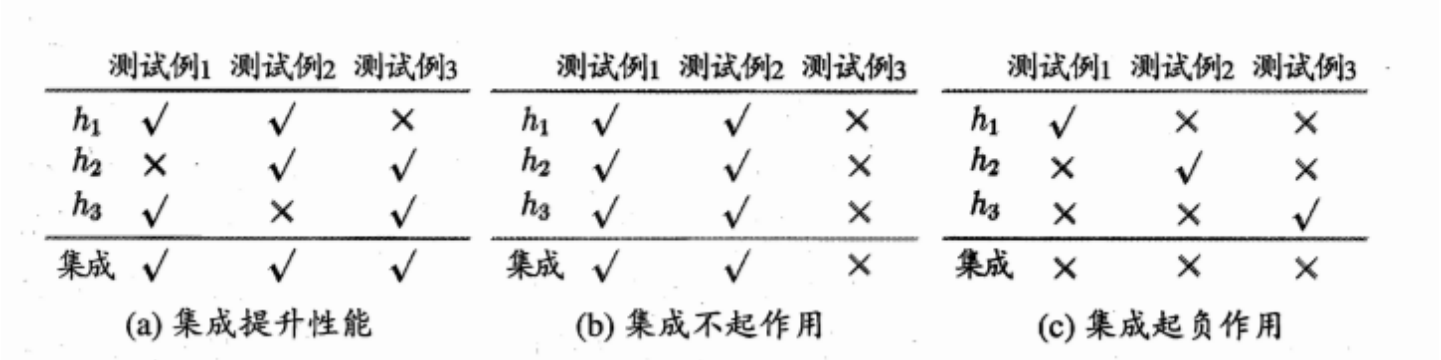

Figure 3: 集成的个体应“好而不同”

5 聚合算法

聚合算法，即 Bagging（**B**ootstrap **agg**regat**ing**）算法，又称装袋算法。聚合算法可与其他分类、回归算法结合，提高其准确率、稳定性的同时，通过降低结果的方差，避免过拟合的发生。例如，集成多个决策树用于分类，每棵决策树给自己预测的类别投一票，最终模型选择票数最多的类别作为结果。

自助抽样 自助抽样的基本思想是从训练集中有放回的随机抽取样本组成多个数据集，每个数据集的容量与原训练集想同。有放回的自助抽样可以让每个抽样出来的数据集与原数据集大小一样，并且抽取出来的数据集之间相互独立。而无放回的抽样不能使抽取出来的数据集大小不变，并且数据集之间存在依赖关系。因此采用有放回的随机抽样。

聚合算法就是基于自助抽样，对于每一个数据集都单独训练一个模型，最后采用投票法等集成方式聚合成一个大模型。例如聚合决策树算法，原数据集有 N 个样本，使用自助抽样抽取出 M 个新的数据集，那么每个数据集也有 N 个样本。将这 M 个数据集分别训练出一棵决策树，那么共有 M 棵决策树。需要对新样本进行预测时，这 M 棵决策树分别给出自己预测类别，最后采用投票法选出一个类别作为结果。

聚合法的特点 对聚合法来说，模型的不稳定性反而是有利的。基础模型越是不稳定，聚合该类模型得到的潜在提升就越大。可变性小的方法（例如 SVM）相较于可变性大的方法（例如决策树），提升是比较小的。

例 5.1. 用数据模拟聚合算法。有 $N = 30$ 个训练样本，每个样本有 $p = 5$ 个特征，样本可以被分为两类。每个特征都服从标准正态分布，相互之间的协方差为 0.95。类别 Y 根据以下规则生成：$P(Y = 1|x_1 \leq 0.5) = 0.2$，$P(Y = 1|x_1 > 0.5) = 0.8$。测试集包含 2000 个样本，自助抽样 $B = 200$ 个训练集，训练决策树模型。下图展示了几棵训练出来的决策树，其中第一张图为使用原训练集训练出来的决策树。

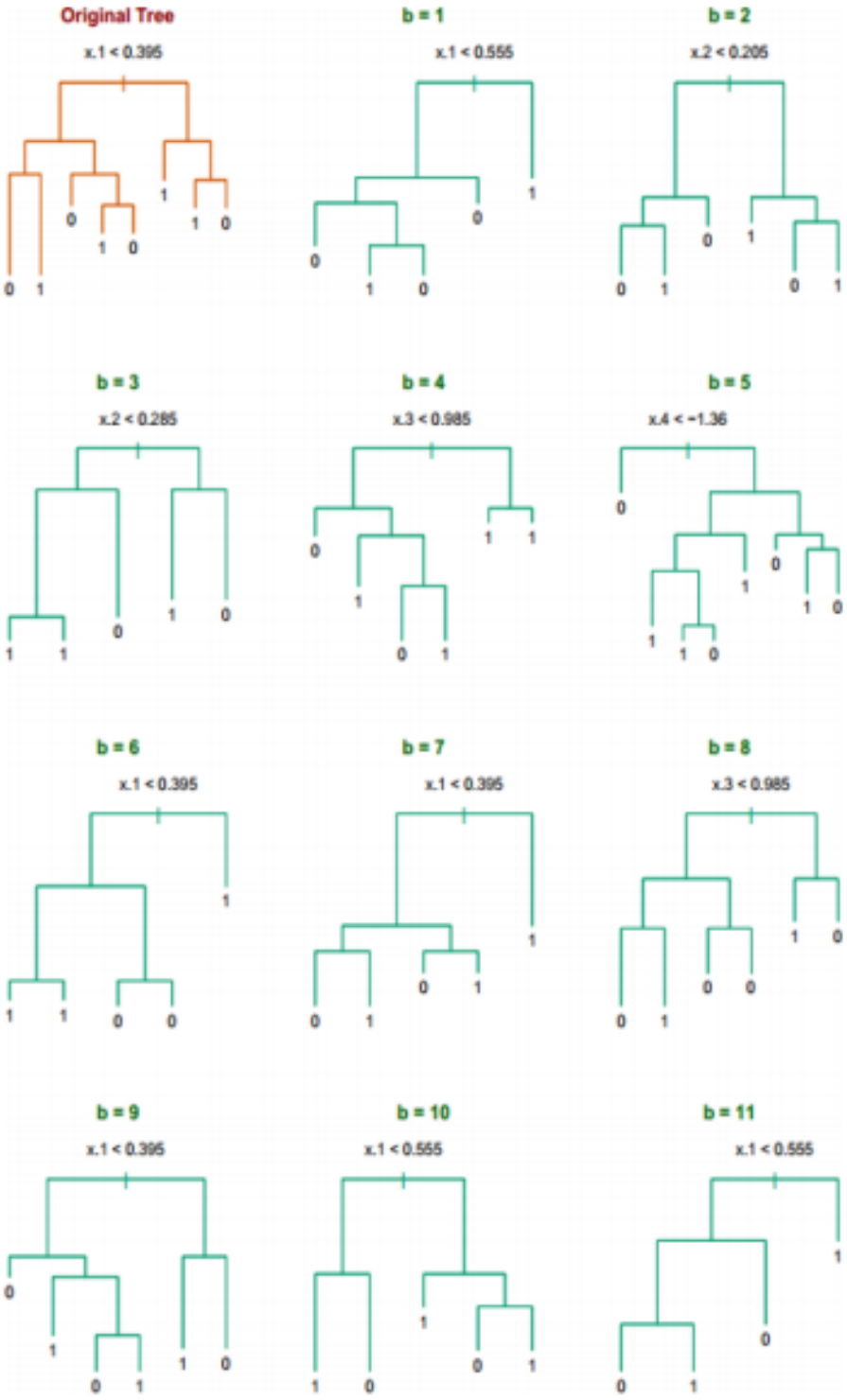

Figure 4: 训练出来的部分决策树

可以发现，用抽样出来的数据集训练的决策树与原数据集训练的树并不是一样的。因为五个特征的相互关联性很高，所有挑选哪个特征作为分裂节点并没有带来什么明显的差异，训练集之间很小的差异也会导致决策树的不同。但是，这些树实际上在分类问题中的表现是非常相似的。图5显示了随着集成个数变化的测试误差，其中 *Consensus* 表示用投票法得出的结果，*Probability* 表示平均概率。不难发现，当 $B > 30$ 后，再增加树的个数已经不能再降低误差，这是因为这些树的相关性很高，给出的预测结果多为相似。

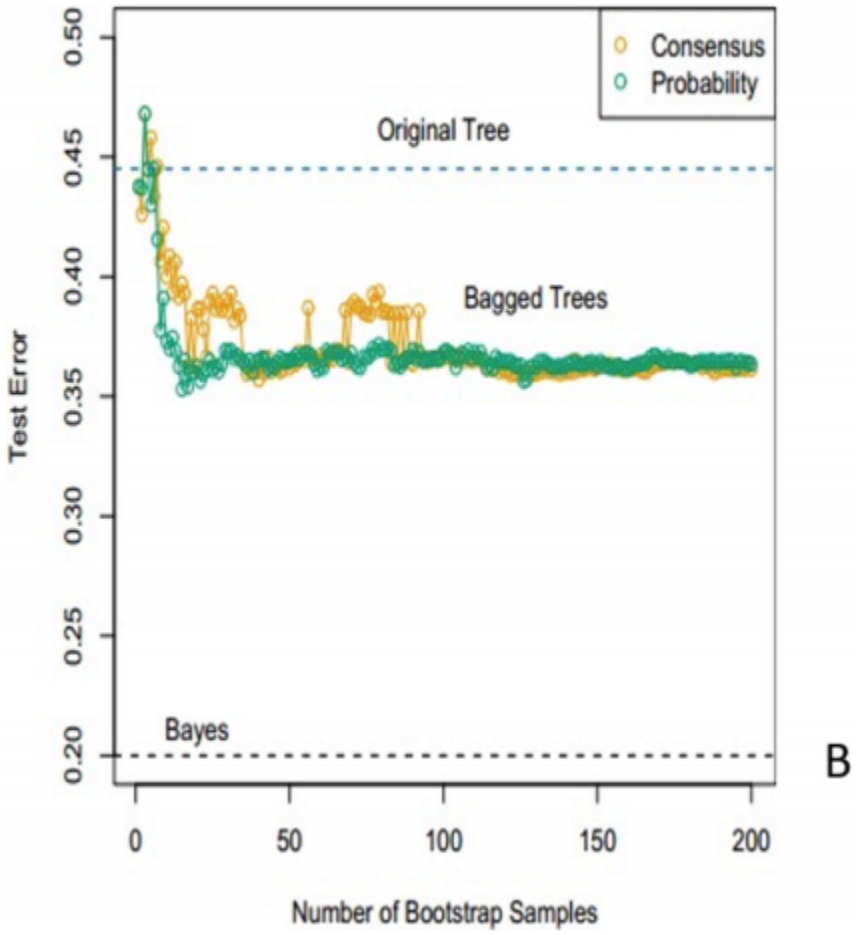

Figure 5: 测试误差

当然聚合算法也有一些缺点，比如模型占用的空间会更大。对于聚合的决策树来说，这些决策树相关性比较大，对应集成学习中的多样性较小，这是不利的。为了减小树之间的相关性，可以使用之后会介绍的随机森林算法。

6 决策树编程实现

我们这里说明怎么用 Python 中的 scikit-learn 库实现一个简单的决策树。

```python
from sklearn import tree
from sklearn.datasets import make_classification
import graphviz

# 生成训练数据，有3个特征，2个类别，共100个样本
X, y = make_classification(n_features=3, n_classes=2, n_redundant=0, n_clusters_per_class=1)

# 训练决策树
clf = tree.DecisionTreeClassifier()
clf.fit(X, y)

# 使用graphviz画出决策树
dot_data = tree.export_graphviz(clf, out_file=None,
                feature_names=['x1', 'x2', 'x3'],
                class_names=['0', '1'],
                filled=True,
                proportion=True)
graph = graphviz.Source(dot_data)
graph.render("decision_tree")
```

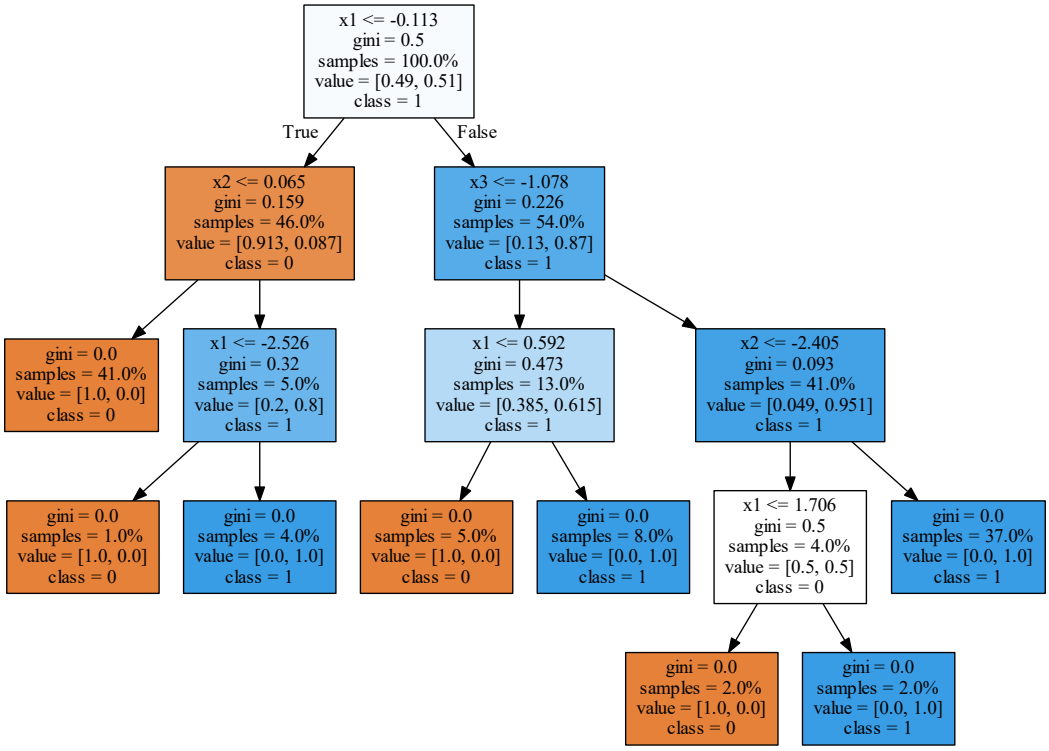

Figure 6: 构建的决策树

引用

[1] https://www.cnblogs.com/zongfa/p/9304353.html

[2] https://blog.csdn.net/perfect1t/article/details/83684995