# 第16课：决策树——既能分类又能回归的模型

一棵决策树（Decision Tree）是一个树结构（可以是二叉树或非二叉树），每个非叶节点对应一个特征，该节点的每个分支代表这个特征的一个取值，而每个叶节点存放一个类别或一个回归函数。

使用决策树进行决策的过程就是从根节点开始，提取出待分类项中相应的特征，按照其值选择输出分支，依次向下，直到到达叶子节点，将叶子节点存放的类别或者回归函数的运算结果作为输出（决策）结果。

决策树的决策过程非常直观，容易被人理解，而且运算量相对小。它在机器学习当中非常重要。如果要列举“十大机器学习模型”的话，决策树应当位列前三。

## 构建决策树

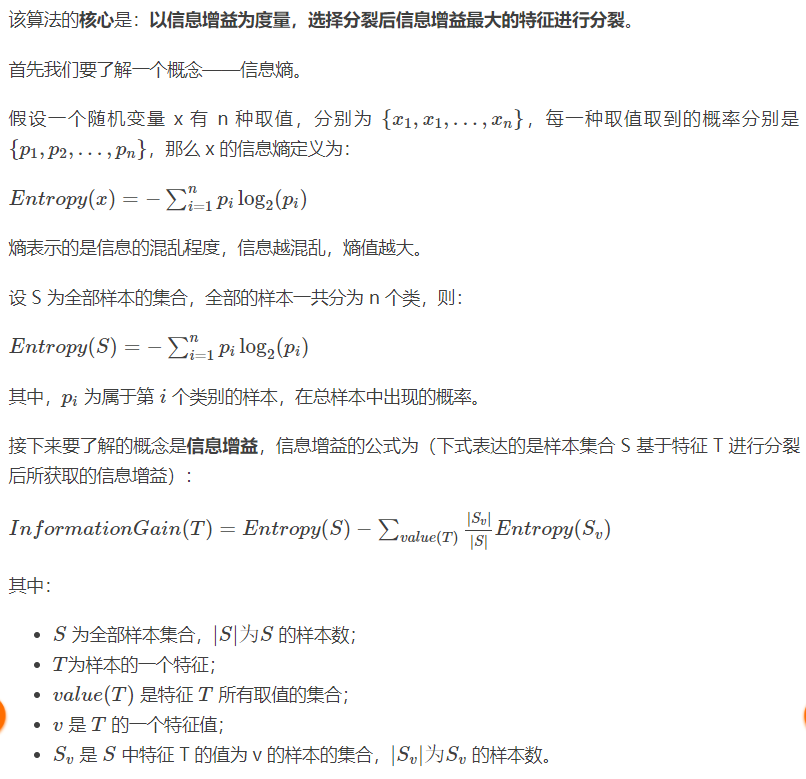
1. 准备若干的训练数据（假设 m 个样本）；
2. 标明每个样本预期的类别；
3. 人为选取一些特征（即决策条件）；
4. 为每个训练样本对应所有需要的特征生成相应值——数值化特征；
5. 将通过上面的1-4步获得的训练数据输入给训练算法，训练算法通过一定的原则，决定各个特征的重要性程度，然后按照决策重要性从高到底，生成决策树。

## 常用算法

决策树的构造过程是一个迭代的过程。每次迭代中，采用不同特征作为分裂点，来将样本数据划分成不同的类别。被用作分裂点的特征叫做**分裂特征**。

选择分裂特征的**目标**，是让各个分裂子集尽可能地“纯”，即尽量让一个分裂子集中的样本都属于同一类别。

### ID3 算法

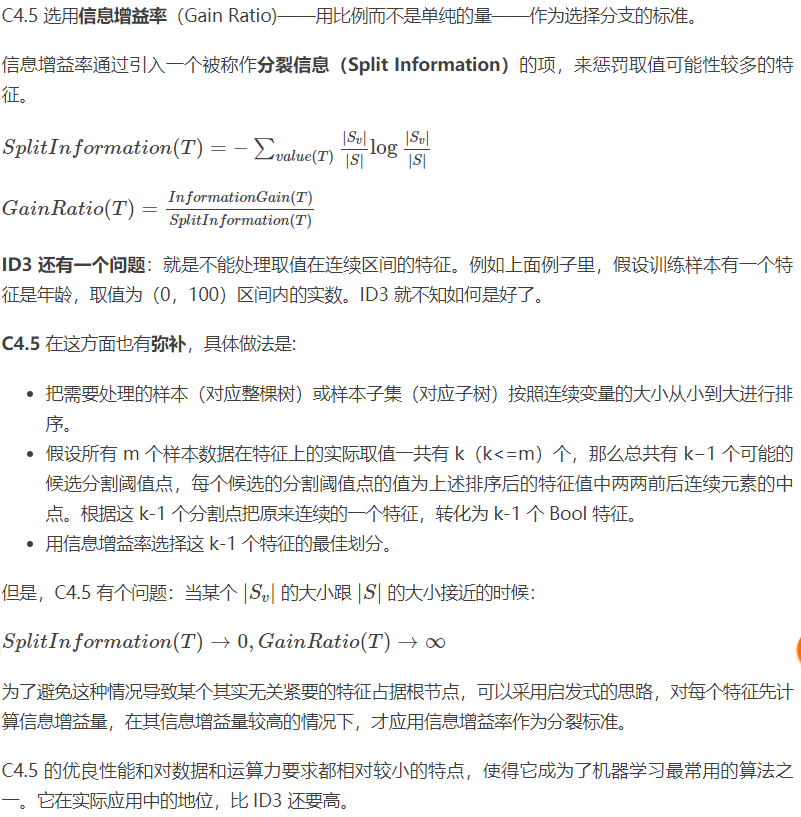


### C4.5

ID3一般会优先选择取值种类较多的特征作为分裂特征。因为取值种类多的特征会有相对较大的信息增益——信息增益反映的是给定一个条件以后不确定性被减少的程度，必然是分得越细的数据集确定性更高。

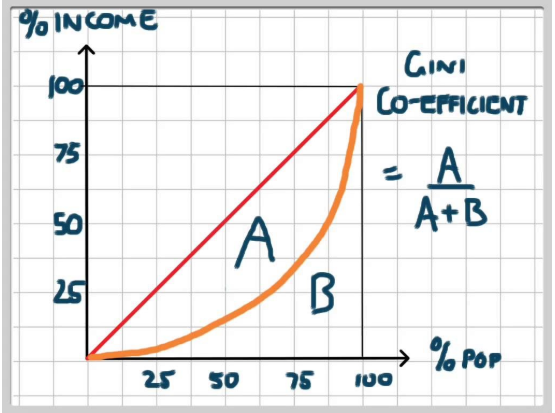
被取值多的特征分裂，分裂成的结果也就容易细；分裂结果越细，则信息增益越大。

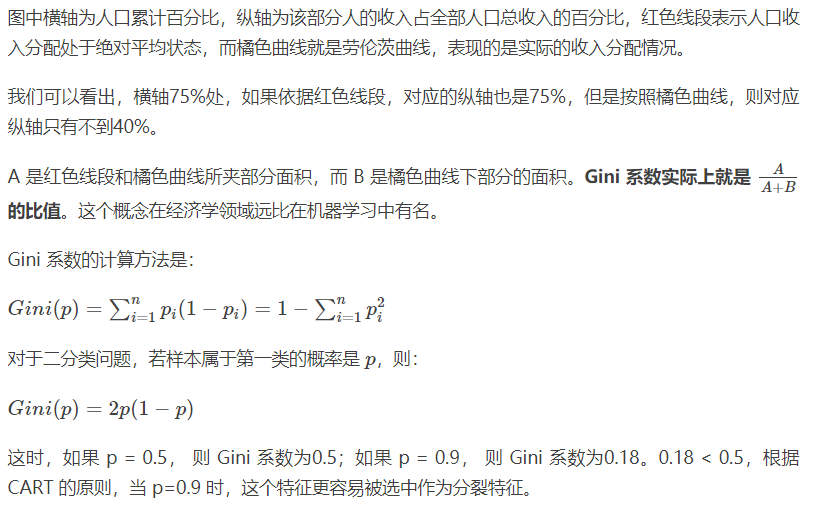
为了避免这个不足，在 ID3 算法的基础上诞生了它的改进版本：C4.5 算法。



### CART







由此可见，对二分类问题中，两种可能性的概率越不平均，则越可能是更佳优越的切分点。

上面的例子虽然用的是二分类，但实际上，对于多分类，趋势是一样的，那些概率分布在不同可能性之间越不平均的特征，越容易成为分裂特征。

到了这里，可能有朋友会误会，认为我们一直说的都是用 CART 做分类时的做法。但是实际上，无论是做分类还是做回归，都是一样的。

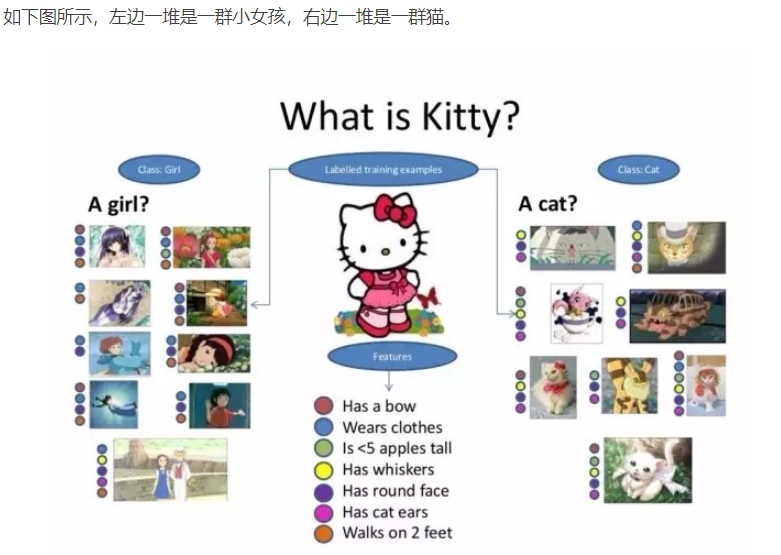
* 回归树和分类树的区别在于最终的输出值到底是连续的还是离散的，每个特征——也就是分裂点决策条件——无论特征值本身是连续的还是离散的，都要被当作离散的来处理，而且都是被转化为二分类特征，来进行处理：
* 如果对应的分裂特征是连续的，处理与 C4.5 算法相似；

如果特征是离散的，而该特征总共有 k 个取值，则将这一个特征转化为 k 个特征，对每一个新特征按照是不是取这个值来分 Yes 和 No。

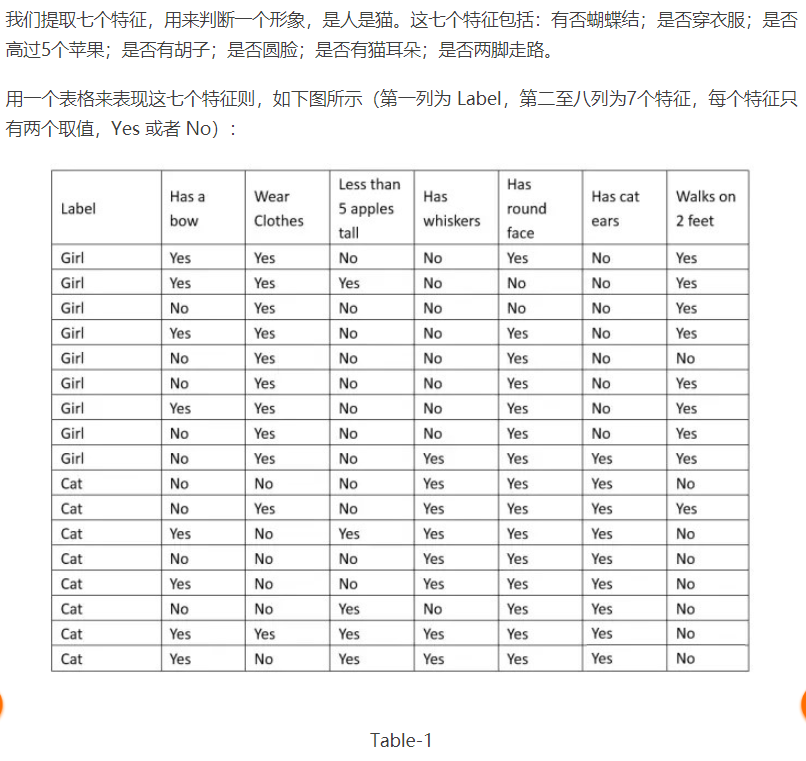
**注意：**还有一个词——Gini 指数（Gini Index），经常在一些资料中被提及，并在 CART 算法中用来代替 Gini 系数，其实 Gini 指数就是基尼系数乘100倍作百分比表示，两者其实是一个东西。

# 第17课：决策树——告诉你 Hello Kitty 是人是猫

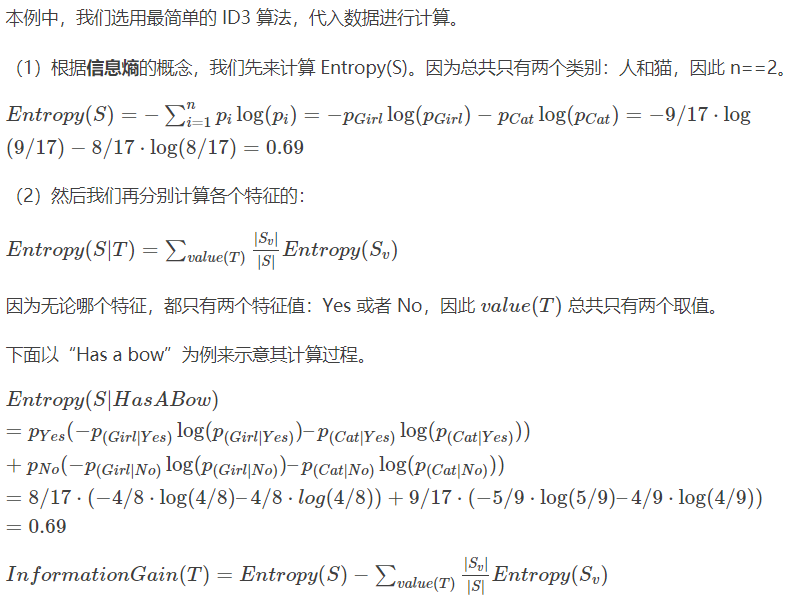
## 训练数据



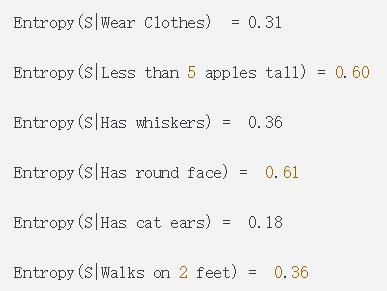
## 特征选取



## 用 ID3 算法构造分类树

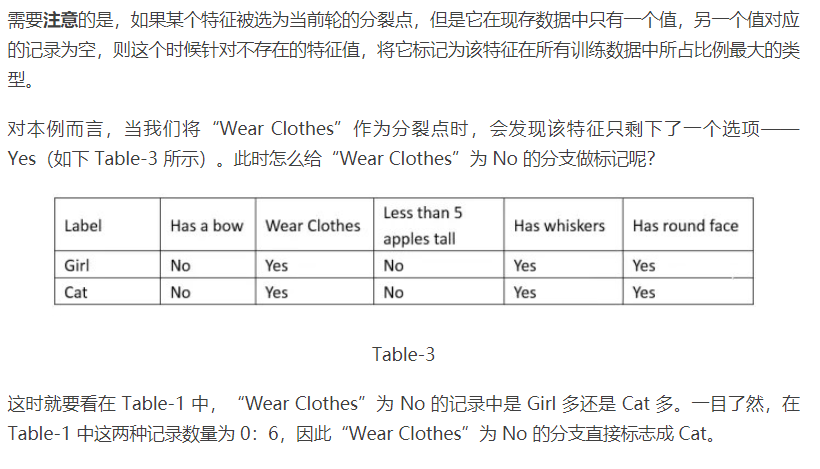


计算结果如下：

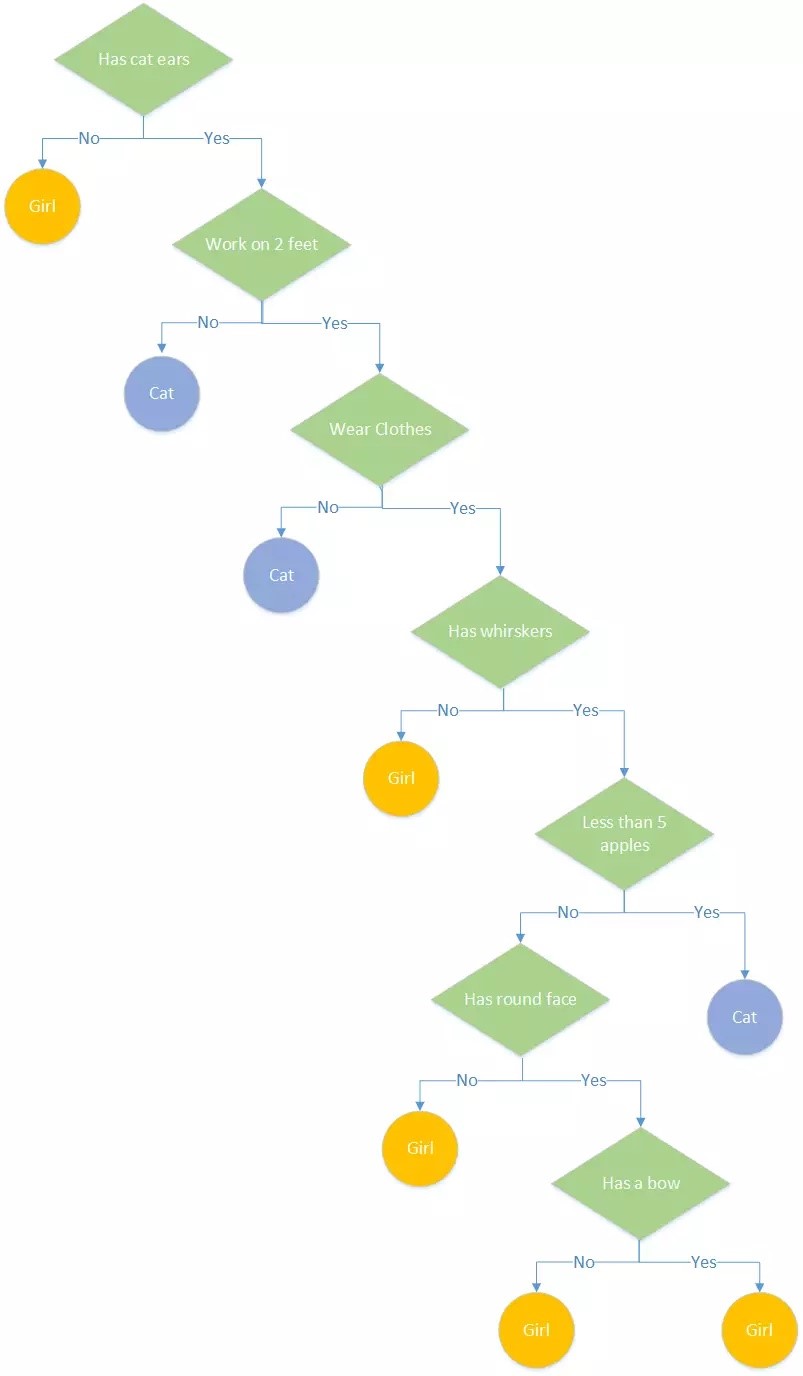








最终决策树如下：



代码如下：

DecisionTree induceTree(training\_set, features) {

If(training\_set中所有的输入项都被标记为同一个label){

return 一个标志位该label的叶子节点；

} else if(features为空) {

# 默认标记为在所有training\_set中所占比例最大的label

return 一个标记为默认label的叶子节点；

} else {

选取一个feature，F；

以F为根节点创建一棵树currentTree；

从Features中删除F；

foreach(value V of F) {

将training\_set中feature F的取值为V的元素全部提取出来，组成partition\_v；

branch\_v= induceTree(partition\_V, features);

将branch\_v添加为根节点的子树，根节点到branch\_v的路径为F的V值；

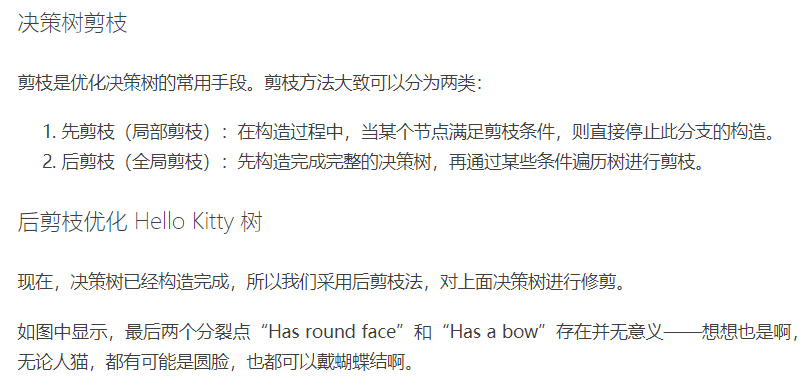
}

returncurrentTree；

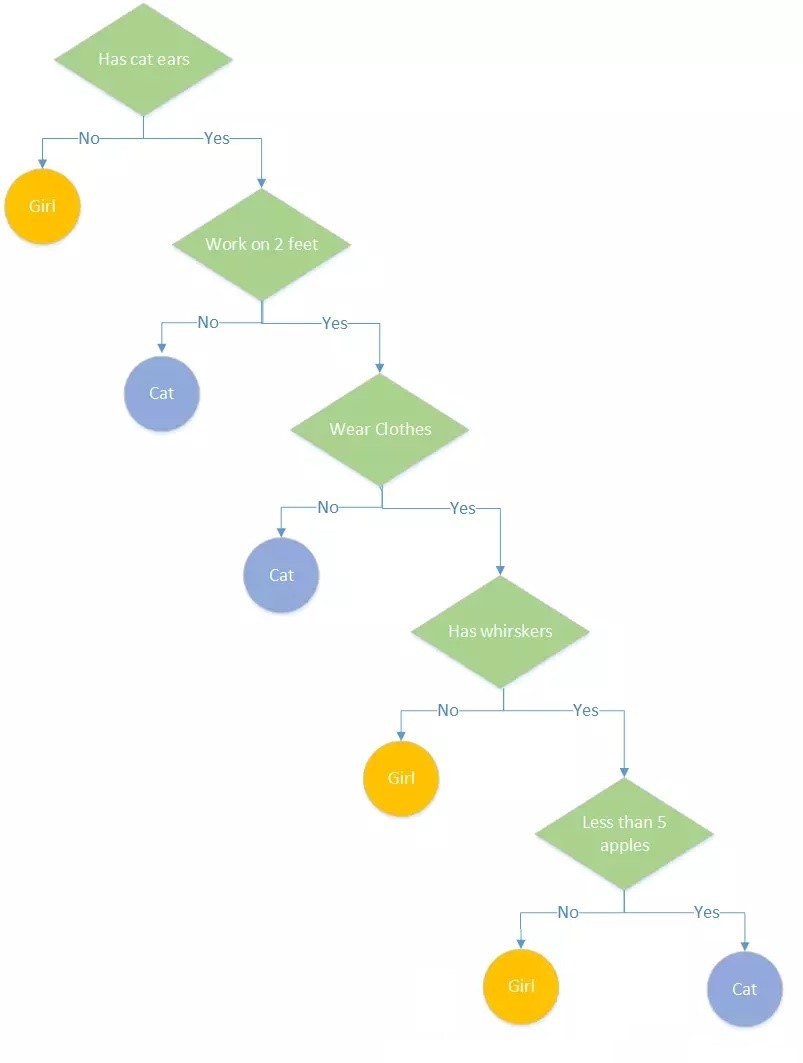
}

}

## 后剪枝优化决策树



最后决策树变成：



## 用决策树对 Hello Kitty 进行分类

我们将 Hello Kitty 的特征带入 Cat-Girl 决策树，发现 Hello Kitty：Has cat ears: Yes -> Work on 2 feed: Yes -> Wear Clothes: Yes -> Has whirskers: Yes -> Less than 5 apples: Yes -> Cat。

Bingo! Hello Kitty 是只猫！这是我们的 ID3 决策树告诉我们的！

代码：（用 numpy 和 sklearn 来实现例子中的训练分类树来判断 Hello Kitty 种族所对应的程序。）

from sklearn import tree

from sklearn.model\_selection im

port train\_test\_split

import numpy as np

#9个女孩和8只猫的数据，对应7个feature，yes取值为1，no为0

features = np.array([

[1, 1, 0, 0, 1, 0, 1],

[1, 1, 1, 0, 0, 0, 1],

[0, 1, 0, 0, 0, 0, 1],

[1, 1, 0, 0, 1, 0, 1],

[0, 1, 0, 0, 1, 0, 0],

[0, 1, 0, 0, 1, 0, 1],

[1, 1, 0, 0, 1, 0, 0],

[0, 1, 0, 0, 1, 0, 1],

[0, 1, 0, 1, 1, 1, 1],

[1, 0, 1, 1, 1, 1, 0],

[0, 0, 0, 1, 1, 1, 0],

[1, 0, 1, 1, 1, 1, 0],

[0, 0, 0, 1, 1, 1, 0],

[1, 0, 0, 1, 1, 1, 0],

[0, 0, 1, 0, 1, 1, 0],

[1, 1, 1, 1, 1, 1, 0],

[1, 0, 1, 1, 1, 1, 0]

])

#1 表示是女孩，0表示是猫

labels = np.array([

[1],

[1],

[1],

[1],

[1],

[1],

[1],

[1],

[1],

[0],

[0],

[0],

[0],

[0],

[0],

[0],

[0],

])

# 从数据集中取20%作为测试集，其他作为训练集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

features,

labels,

test\_size=0.2,

random\_state=0,

)

# 训练分类树模型

clf = tree.DecisionTreeClassifier()

clf.fit(X=X\_train, y=y\_train)

# 测试

print(clf.predict(X\_test))

# 对比测试结果和预期结果

print(clf.score(X=X\_test, y=y\_test))

# 预测HelloKitty

HelloKitty = np.array([[1,1,1,1,1,1,1]])

print(clf.predict(HelloKitty))