决策树实战

每个标题对应notebook文件，同学们可以自己运行一下。

01 What is Decision Tree

我们先来看一个简单的决策树代码，使用scikit-learn里的DecisionTreeClassifier。

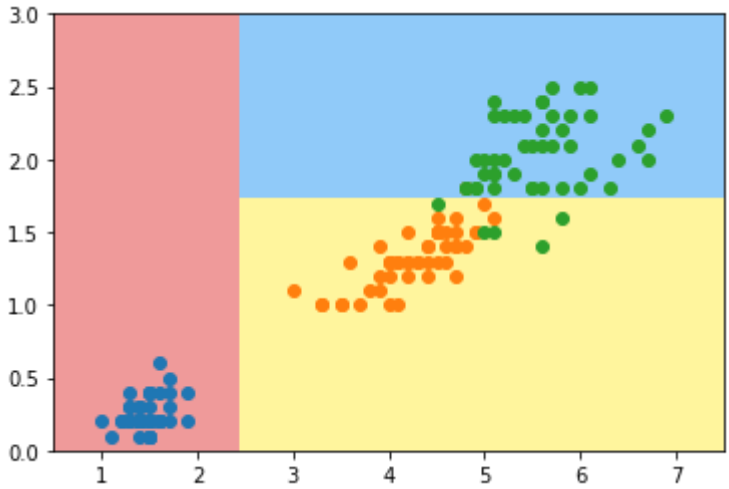
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets

# 加载鸢尾花卉数据集
iris = datasets.load_iris()
X = iris.data[:,2:]
y = iris.target

from sklearn.tree import DecisionTreeClassifier

dt_clf = DecisionTreeClassifier(max_depth=2, criterion="entropy", random_state=42)
dt_clf.fit(X, y)
```

我们还可以绘制决策边界，更直观的了解决策树是怎么划分数据的。



具体代码参照01-What-is-Decision-Tree.ipynb

02 Entropy

决策树又称为判定树，是运用于分类的一种树结构，其中的**每个内部节点代表对某一属性的一次测试**，每条边代表一个测试结果，叶节点代表某个类或类的分布。

那么如何确定**每个节点在哪个特征上做划分？每个特征在哪个值上做划分？**

我们引入信息论中的熵的概念。

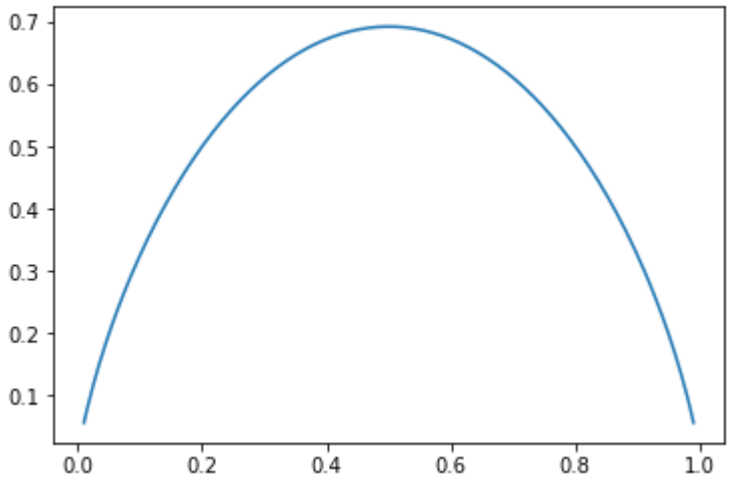
熵在信息学中代表随机变量不确定度的度量。熵越大，数据的不确定性越高。

所以我们可以通过划分前后熵的变化来确定每个节点的特征和每个特征的值的选择。

熵的公式为：

$$H = -\sum_{i=1}^{k} p_i log(p_i)$$

其函数图像为：



03 Entropy Split Simulation

通过最大化节点划分前后熵的差值，就是ID3

```
from collections import Counter
from math import log

# 信息熵计算
def entropy(y):
    counter = Counter(y)
    res = 0.0
    for num in counter.values():
        p = num / len(y)
        res += -p * log(p)
    return res

# 找到最大的信息熵差值，最优的划分特征和最优的划分边界
def try_split(X, y):
    best_entropy = float('inf')
    best_d, best_v = -1, -1
    for d in range(X.shape[1]):
        sorted_index = np.argsort(X[:,d])
        for i in range(1, len(X)):
            if X[sorted_index[i], d] != X[sorted_index[i-1], d]:
                v = (X[sorted_index[i], d] + X[sorted_index[i-1], d])/2
                X_l, X_r, y_l, y_r = split(X, y, d, v)
                e = entropy(y_l) + entropy(y_r)
```

```
                if e < best_entropy:
                    best_entropy, best_d, best_v = e, d, v

    return best_entropy, best_d, best_v
```

04 Gini Index

基尼系数代表了模型的纯度，基尼系数越小，则纯度越高，特征越好。

具体的，在分类问题中，假设有K个类别，第k个类别的概率为p_k，则基尼系数的表达式为：

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

gini系数的代码实现：

```
def gini(y):
    counter = Counter(y)
    res = 1.0
    for num in counter.values():
        p = num / len(y)
        res -= p**2
    return res

def try_split(X, y):
    best_g = float('inf')
    best_d, best_v = -1, -1
    for d in range(X.shape[1]):
        sorted_index = np.argsort(X[:,d])
        for i in range(1, len(X)):
            if X[sorted_index[i], d] != X[sorted_index[i-1], d]:
                v = (X[sorted_index[i], d] + X[sorted_index[i-1], d])/2
                X_l, X_r, y_l, y_r = split(X, y, d, v)
                g = gini(y_l) + gini(y_r)
                if g < best_g:
                    best_g, best_d, best_v = g, d, v

    return best_g, best_d, best_v
```

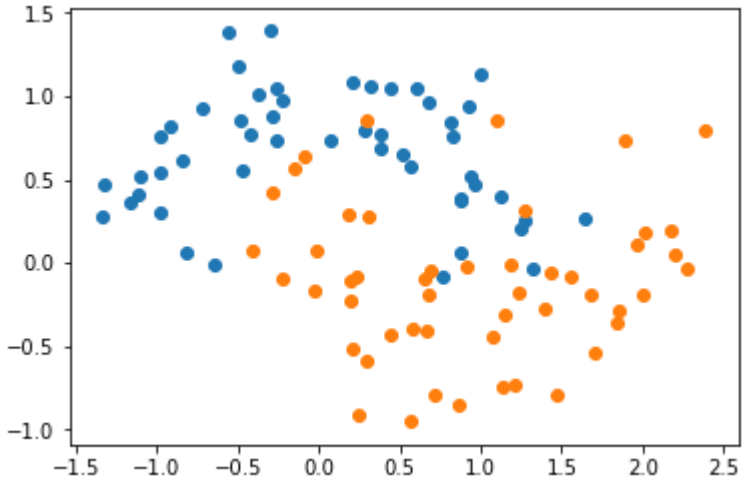
信息熵vs基尼系数

- 信息熵计算较慢（涉及大量对数运算）
- scikit-learn中默认为基尼系数
- 大多数时候二者没有特别的效果优劣

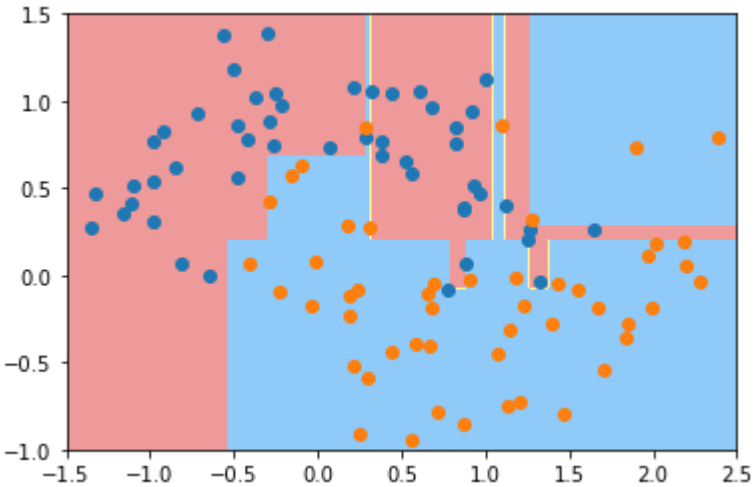
05 CART and Decisioin Tree Hyperparameters

在这里讲一下skitit learn中DecisionTreeClassifier（默认为CART）的超参数。使用sklearn中的moons数据集。

数据集分布：

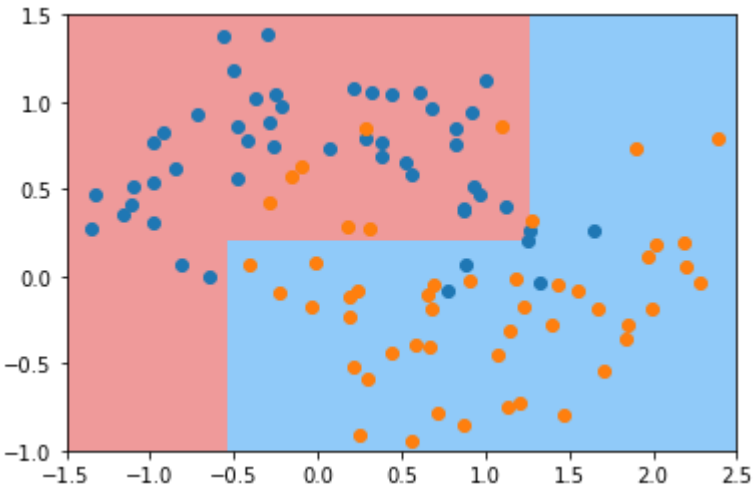


不传超参数时的决策边界：

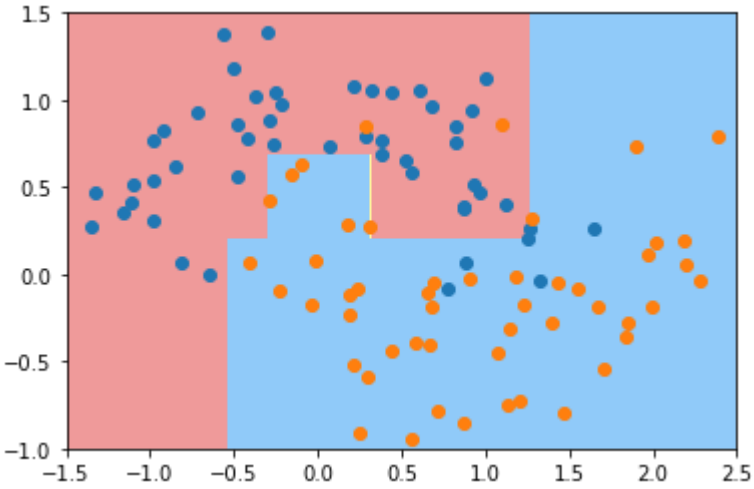


可以看到过拟合的情况发生。

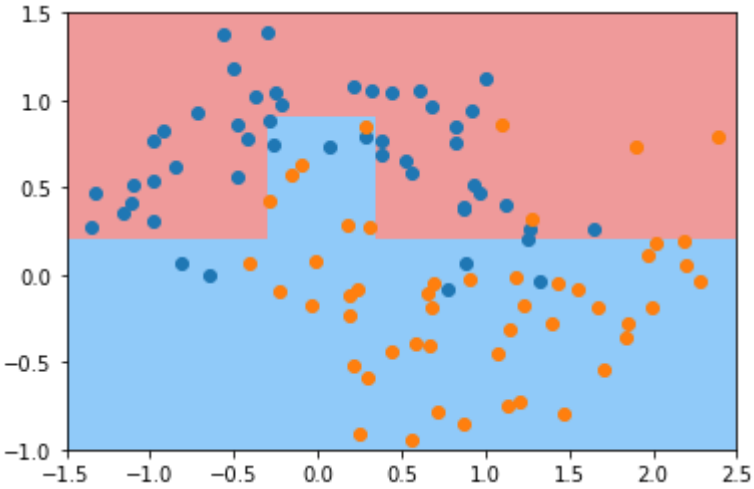
当我们加了超参数max_depth=2时，我们可以看到效果好了很多：



min_samples_split=10，内部节点再划分所需最小样本数。默认值为2。显而易见，节点再划分所需的样本数提高，可以增加泛化能力，对应的划分结果（与第一张不传超参数的对比）：



min_samples_leaf=6，如果您以前编写过一个决策树，你能体会到最小样本叶片大小的重要性。 叶是决策树的末端节点。 较小的叶子使模型更容易捕捉训练数据中的噪声。 一般来说，我更偏向于将最小叶子节点数目设置为大于50。在你自己的情况中，你应该尽量尝试多种叶子大小种类，以找到最优的那个。对应的结果为：



max_leaf_nodes=4，通过限制最大叶子节点数，可以防止过拟合，默认是"None”，即不限制最大的叶子节点数。如果加了限制，算法会建立在最大叶子节点数内最优的决策树。如果特征不多，可以不考虑这个值，但是如果特征分成多的话，可以加以限制，具体的值可以通过交叉验证得到。