第8章 决策树（Decision Tree）

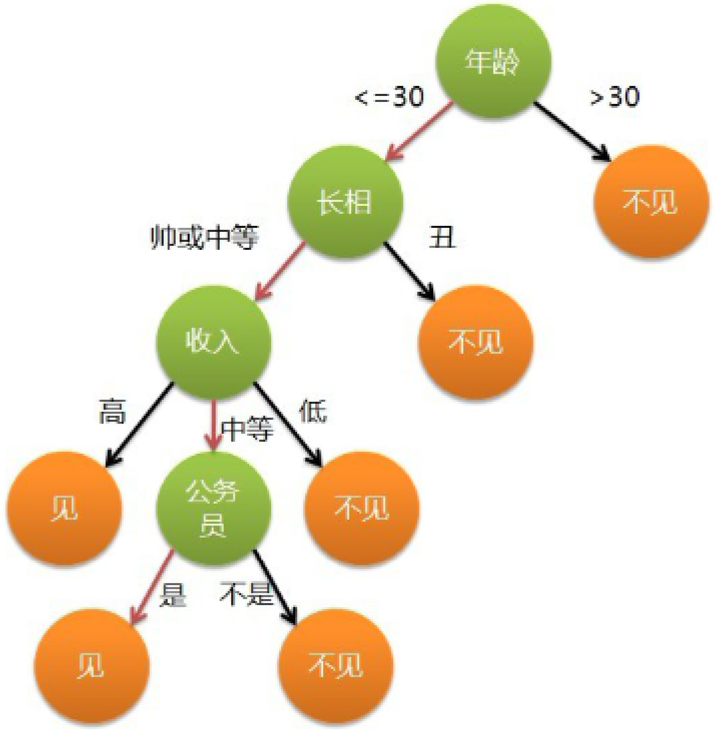
[1]

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn import datasets
from sklearn.model_selection import train_test_split
from collections import Counter
from math import log

import sys
import os

sys.path.append('.')
sys.path.append('..')
from mlUtils.plot_decision_boundary import plot_decision_boundary
```

一、算法思想



二、信息熵与基尼系数

1．信息熵

假设有K个分类，每个分类点的概率为：$p_i = P(X = x_i)$。则表示信息增益的信息熵定义如下。

$$H = -\sum_{i=1}^{K} p_i \log(p_i) \ (0 < p_i \leq 1)$$

特别的，如果是二分类，我们可以将上述公式写为：

$$H = -\left[p \log p + (1-p) \log(1-p)\right]$$

- 分类1：$\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\}$

[3]

```
H1 = -1/3*log(1/3)*3
print(H1)
```

```
1.0986122886681096
```

- 分类2：$\{\frac{1}{10}, \frac{1}{10}, \frac{4}{5}\}$

[4]

```
H2 = -1/10*log(1/10)*2-4/5*log(4/5)
print(H2)
```

0.639031859650177

[7]

```
H3 = log(1)
print(H3)
```

0.0

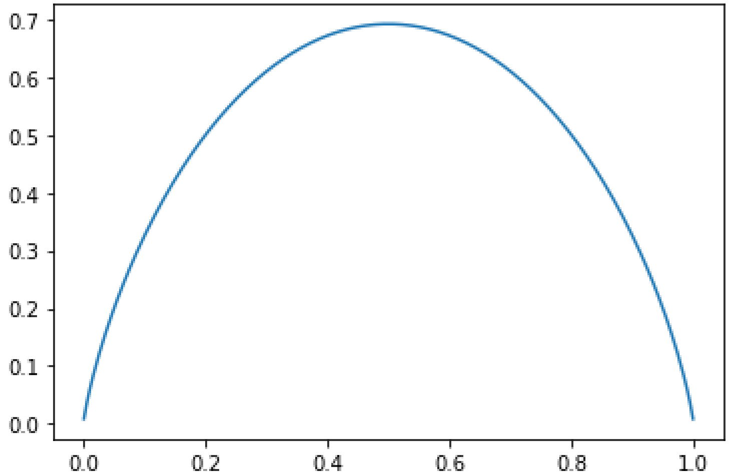
- 分类3： $\{1, 0, 0\}$

[8]

```
def entropy_2(p):
    return -p*np.log(p) - (1-p)*np.log(1-p)
```

[9]

```
x = np.linspace(0.001, 0.999, 200)
plt.plot(x, entropy_2(x))
plt.show()
```



2．基尼系数

$$G = 1 - \sum_{i=1}^{K} p_i^2$$

特别的，如果是二分类，我们可以将上述公式写为：

$$G = 1 - \left[p^2 + (1-p)^2\right] = -2p^2 + 2p$$

3．两种指标比较

- 基尼系数速度稍微
- 约大多数情况下两者差别比较小

三、决策树的实现（基于信息熵）

1．导入数据

[2]
```python
iris = datasets.load_iris()
X = iris.data[:, 2:]
y = iris.target
```

2. SKLearn中定义分类器并训练

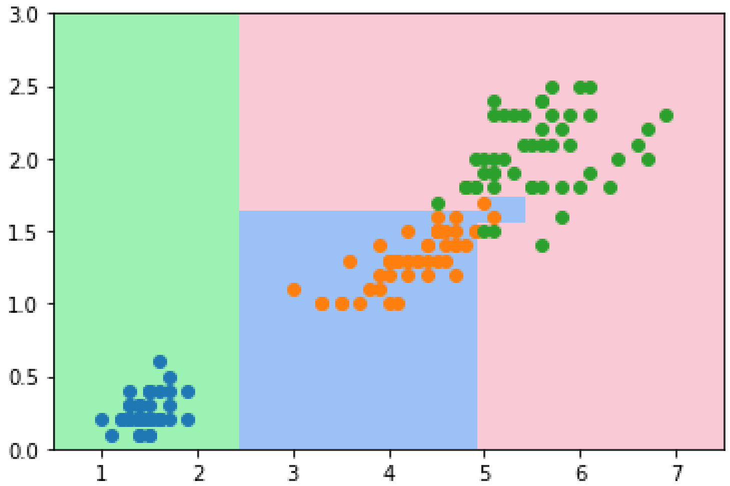
[3]
```python
X = iris.data[:, 2:]
y = iris.target
dt_clf_entropy = DecisionTreeClassifier(max_depth=5,
criterion="entropy")
dt_clf_entropy.fit(X, y)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=5)
```

3. 绘制决策边界

[4]
```python
plot_decision_boundary(dt_clf_entropy, axis=[0.5, 7.5, 0, 3])
plt.scatter(X[y==0, 0], X[y==0, 1])
plt.scatter(X[y==1, 0], X[y==1, 1])
plt.scatter(X[y==2, 0], X[y==2, 1])
```

```python
plt.show()
```



4. 手工分支（信息熵）

[5]

```python
# X shape: n*m, y: 1*m
def split(X, y, dim, sValue):
    index_le = (X[dim, :] <= sValue)
    index_gr = (X[dim, :] > sValue)
    return X[:, index_le], X[:, index_gr], y[index_le], y[index_gr]
```

[6]

```python
def entorpy_m(y):
    h = 0.0
    counter = Counter(y)
    for subItems in counter.values():
```

```
        p = subItems / len(y)
        h += -p * log(p)

    return h
```

[7]

```
def try_split(X, y):
    best_entropy = float("inf")
    best_dim, best_value = -1, -1
    for dim in range(X.shape[0]):
        sorted_index = np.argsort(X[dim, :])
        for i in range(1, X.shape[1]):
            if X[dim, sorted_index[i-1]] != X[dim, sorted_index[i]]
:
                sValue = (X[dim, sorted_index[i-1]] + X[dim,
sorted_index[i]])/2
                X_lt, X_rt, y_lt, y_rt = split(X, y, dim, sValue)
                entropy = entorpy_m(y_lt) + entorpy_m(y_rt)
                if entropy < best_entropy:
                    best_entropy, best_dim, best_value = entropy,
dim, sValue

    return best_entropy, best_dim, best_value
```

[8]

```
best_entropy1, best_dim1, best_value1 = try_split(X.T, y)
print("best entropy: ", best_entropy1)
print("best dim: ", best_dim1)
print("best value: ", best_value1)
```

```
best entropy:  0.6931471805599453
```

```
best dim:  0
best value:  2.45
```

[18]

```python
X_lt_1, X_rt_1, y_lt_1, y_rt_1 = split(X.T, y, best_dim1,
best_value1)
```

[19]

```python
(entorpy_m(y_lt_1), entorpy_m(y_rt_1))
```

```
(0.0, 0.6931471805599453)
```

[20]

```python
best_entropy2, best_dim2, best_value2 = try_split(X_rt_1,  y_rt_1)
print("best entropy: ", best_entropy2)
print("best dim: ", best_dim2)
print("best value: ", best_value2)
```

```
best entropy:  0.4132278899361904
best dim:  1
best value:  1.75
```

[21]

```python
X_lt_2, X_rt_2, y_lt_2, y_rt_2 = split(X_rt_1,  y_rt_1, best_dim2,
 best_value2)
```

[22]

```python
(entorpy_m(y_lt_2), entorpy_m(y_rt_2))
```

```
(0.30849545083110386, 0.10473243910508653)
```

四、决策树的实现（基尼系数）

1. SKLearn中基尼决策树

[31]

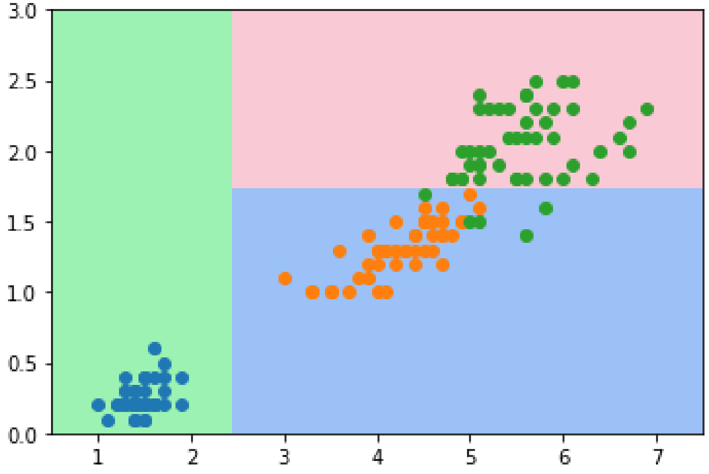
```
X = iris.data[:, 2:]
y = iris.target
dt_clf_gini = DecisionTreeClassifier(max_depth=2,
criterion="gini")
dt_clf_gini.fit(X, y)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=2,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

[32]

```
plot_decision_boundary(dt_clf_gini, axis=[0.5, 7.5, 0, 3])
plt.scatter(X[y==0, 0], X[y==0, 1])
plt.scatter(X[y==1, 0], X[y==1, 1])
plt.scatter(X[y==2, 0], X[y==2, 1])
```

```
plt.show()
```



2. 手工分支（基于基尼系数）

[33]

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

    return g
```

[34]

```
def try_split_gini(X, y):
    best_gini = float("inf")
    best_dim, best_value = -1, -1
```

```
    for dim in range(X.shape[0]):
        sorted_index = np.argsort(X[dim, :])
        for i in range(1, X.shape[1]):
            if X[dim, sorted_index[i-1]] != X[dim, sorted_index[i]
]:
                sValue = (X[dim, sorted_index[i-1]] + X[dim,
sorted_index[i]])/2
                X_lt, X_rt, y_lt, y_rt = split(X, y, dim, sValue)
                g = gini(y_lt) + gini(y_rt)
                if g < best_gini:
                    best_gini, best_dim, best_value = g, dim,
sValue

    return best_gini, best_dim, best_value
```

[35]

```
X_best_gini1, best_dim1, best_value1 = try_split_gini(X.T, y)
print("best gini: ", X_best_gini1)
print("best dim: ", best_dim1)
print("best value: ", best_value1)
```

```
best gini:  0.5
best dim:  0
best value:  2.45
```

[36]

```
X_lt_1, X_rt_1, y_lt_1, y_rt_1 = split(X.T, y, best_dim1,
best_value1)
```

[37]

```
(gini(y_lt_1), gini(y_rt_1))
```

```
(0.0, 0.5)
```

[38]

```python
X_best_gini2, best_dim2, best_value2 = try_split_gini(X_rt_1,
y_rt_1)
print("best gini: ", X_best_gini2)
print("best dim: ", best_dim2)
print("best value: ", best_value2)
```

```
best gini:  0.2105714900645938
best dim:  1
best value:  1.75
```

[39]

```python
X_lt_2, X_rt_2, y_lt_2, y_rt_2 = split(X_rt_1, y_rt_1, best_dim2,
best_value2)
```

[40]

```python
(gini(y_lt_2), gini(y_rt_2))
```

```
(0.1680384087791495, 0.04253308128544431)
```

五、其它

1. 分支指标

- CART（Classification And Regression Tree, SKLearn）
- ID3
- C4.5
- C5.0

2. 复杂度

- 预测：O(log(m))
- 训练：O(*n*mlog(m)

3. 剪枝

- 降低复杂度
- 减少过拟合

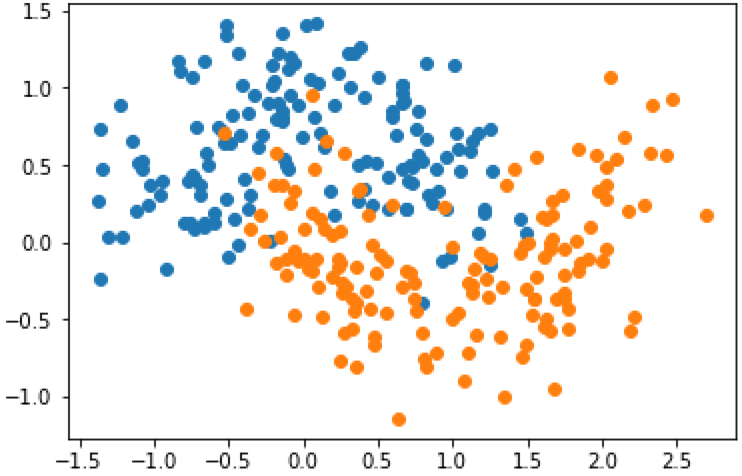
4. 其它参数

[41]

```
X_new, y_new = datasets.make_moons(noise=0.3, random_state=12, n_samples=300)
```

[42]

```
plt.scatter(X_new[y_new==0, 0], X_new[y_new==0, 1])
plt.scatter(X_new[y_new==1, 0], X_new[y_new==1, 1])
plt.show()
```



[43]

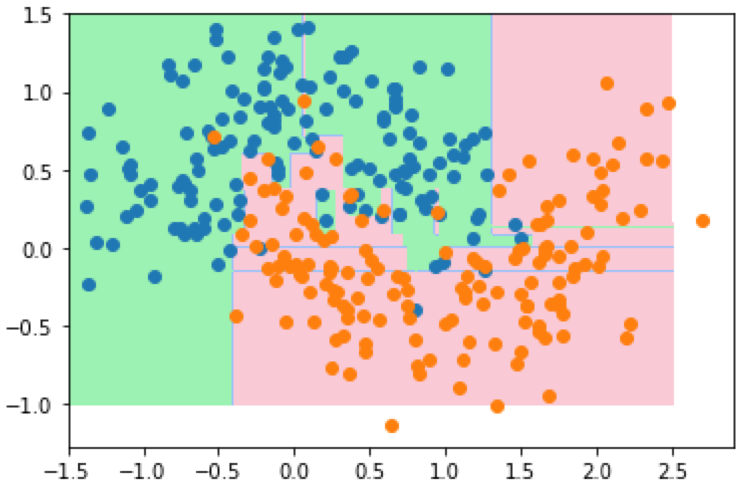
```
dt_clf_moon = DecisionTreeClassifier()
dt_clf_moon.fit(X_new, y_new)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

[44]

```
plot_decision_boundary(dt_clf_moon, axis=[-1.5, 2.5, -1.0, 1.5])
plt.scatter(X_new[y_new==0, 0], X_new[y_new==0, 1])
```

```
plt.scatter(X_new[y_new==1, 0], X_new[y_new==1, 1])
plt.show()
```



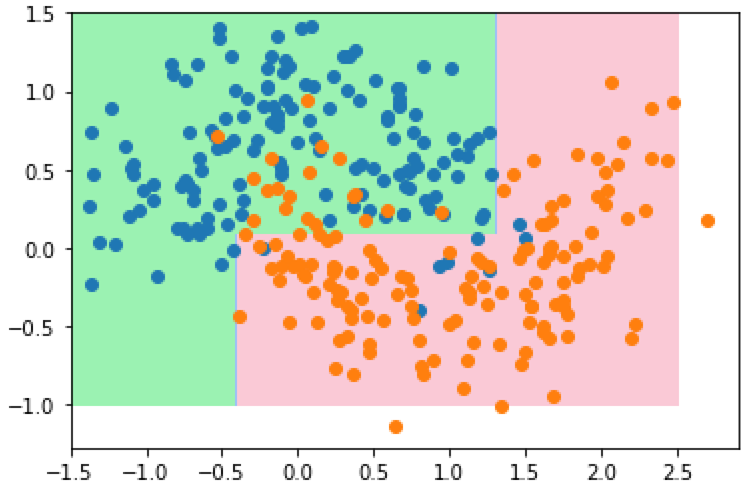
[45]

```
dt_clf_1 = DecisionTreeClassifier(max_depth=2)
dt_clf_1.fit(X_new, y_new)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=2,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

[46]

```
plot_decision_boundary(dt_clf_1, axis=[-1.5, 2.5, -1.0, 1.5])
plt.scatter(X_new[y_new==0, 0], X_new[y_new==0, 1])
plt.scatter(X_new[y_new==1, 0], X_new[y_new==1, 1])
plt.show()
```



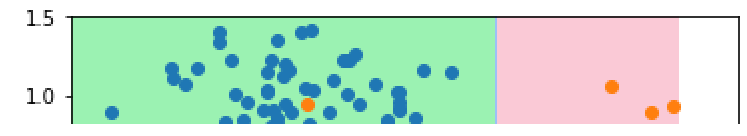
[49]

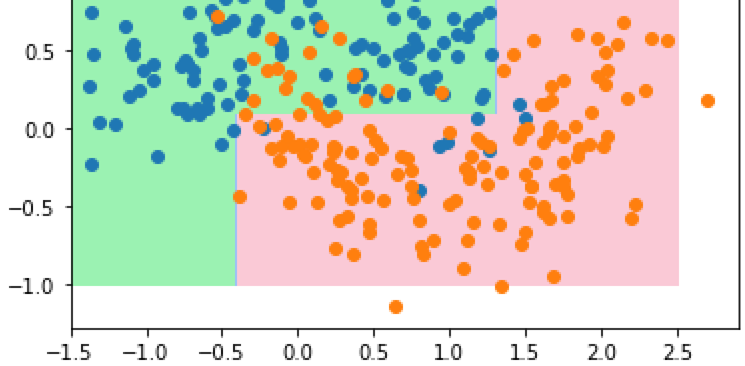
```python
dt_clf_2 = DecisionTreeClassifier(min_samples_split=30)
dt_clf_2.fit(X_new, y_new)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=30,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

[50]

```python
plot_decision_boundary(dt_clf_2, axis=[-1.5, 2.5, -1.0, 1.5])
plt.scatter(X_new[y_new==0, 0], X_new[y_new==0, 1])
plt.scatter(X_new[y_new==1, 0], X_new[y_new==1, 1])
plt.show()
```





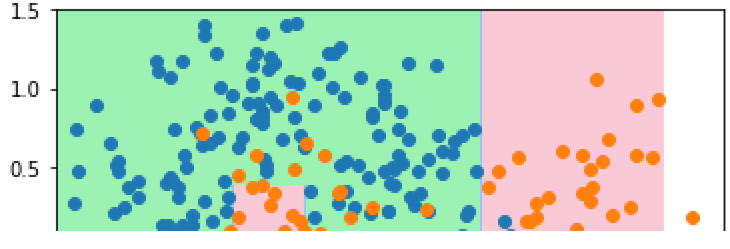
[51]

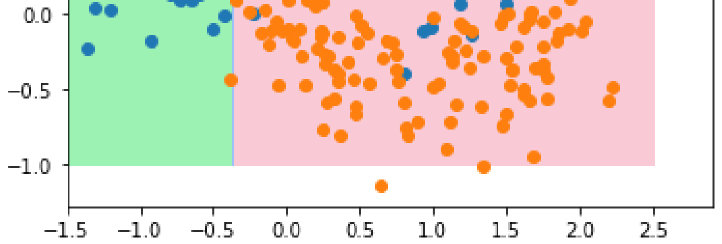
```
dt_clf_3 = DecisionTreeClassifier(min_samples_leaf=7)
dt_clf_3.fit(X_new, y_new)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=7, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

[52]

```
plot_decision_boundary(dt_clf_3, axis=[-1.5, 2.5, -1.0, 1.5])
plt.scatter(X_new[y_new==0, 0], X_new[y_new==0, 1])
plt.scatter(X_new[y_new==1, 0], X_new[y_new==1, 1])
plt.show()
```





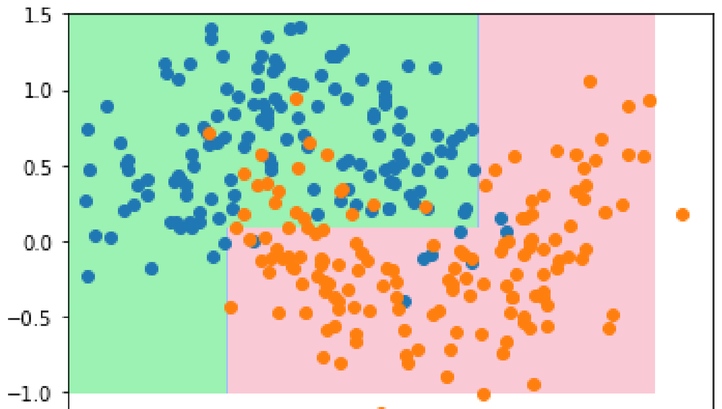
[53]

```
dt_clf_4 = DecisionTreeClassifier(max_leaf_nodes=5)
dt_clf_4.fit(X_new, y_new)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=5,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

[54]

```
plot_decision_boundary(dt_clf_4, axis=[-1.5, 2.5, -1.0, 1.5])
plt.scatter(X_new[y_new==0, 0], X_new[y_new==0, 1])
plt.scatter(X_new[y_new==1, 0], X_new[y_new==1, 1])
plt.show()
```



六、决策树回归

[9]

```
boston = datasets.load_boston()
X_h = boston.data
y_h = boston.target
X_train, X_test, y_train, y_test = train_test_split(X_h, y_h)
```

[10]

```
dt_reg = DecisionTreeRegressor()
dt_reg.fit(X_train, y_train)
```

```
DecisionTreeRegressor()
```

[11]

```
dt_reg.score(X_test, y_test)
```

```
0.7709630203844713
```

[12]

```
dt_reg.score(X_train, y_train)
```

```
1.0
```

[-]