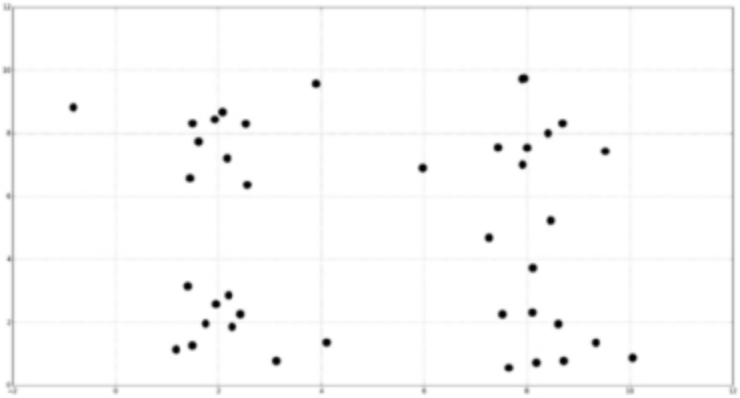
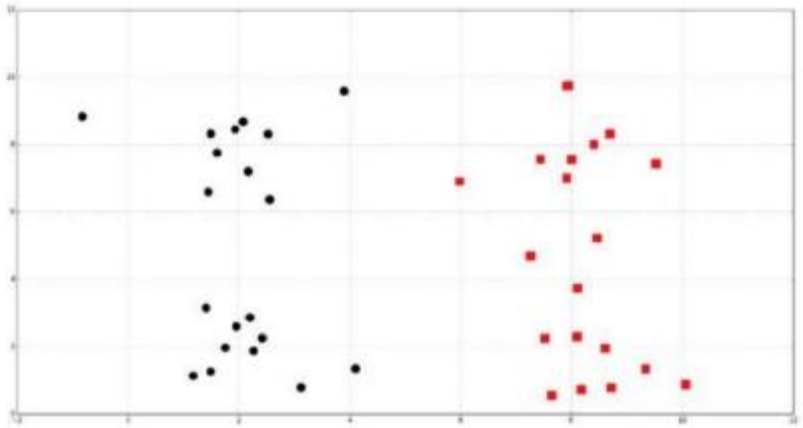
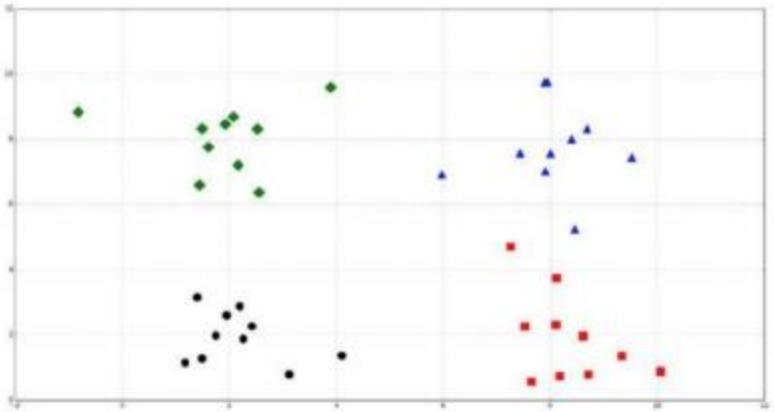
聚类

由聚类所生成的簇是一组数据对象的集合，这些对象与同一个簇中的对象彼此相似，与其他簇中的对象相异







样本点的关键度量指标：距离

- 距离的定义
- 常用距离
  - 欧氏距离，euclidean—通常意义下的距离

$$d_{ij}(2) = \sqrt{\sum_{k=1}^{p}(x_{ik} - x_{jk})^2}.$$

  - 马氏距离，manhattan—考虑到变量间的相关性，并且与变量的单位无关

$$d_{ij}(M) = \sqrt{(x_{(i)} - x_{(j)})^T S^{-1}(x_{(i)} - x_{(j)})},$$

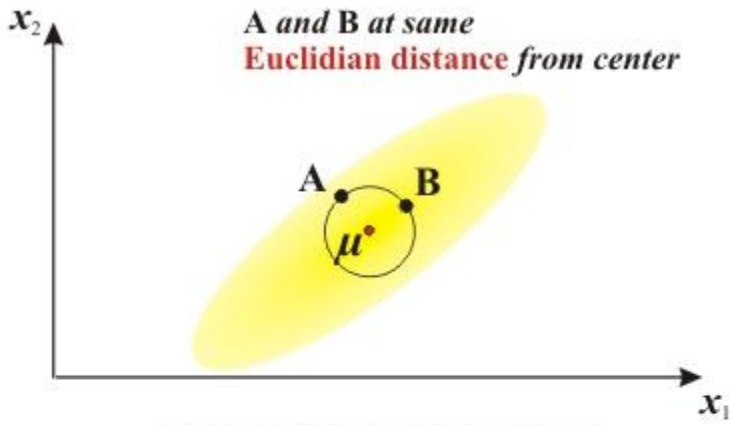
其中 $x_{(i)} = (x_{i1}, x_{i2}, \cdots, x_{ip})^T$, $x_{(j)} = (x_{j1}, x_{j2}, \cdots, x_{jp})^T$, S 为样本方差矩阵.

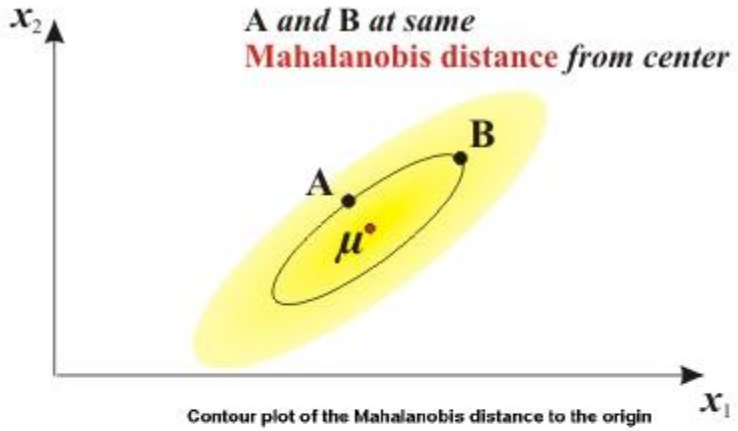
  - 余弦距离，cosine—衡量变量相似性

$$\mathrm{d}_{ij} = \frac{x_1 y_1 + x_2 y_2 + \cdots\ldots + x_n y_n}{\sqrt{x_{12} + x_{22} + \cdots\ldots + x_{n2}} \times \sqrt{y_{12} + y_{22} + \cdots\ldots + y_{n2}}}$$

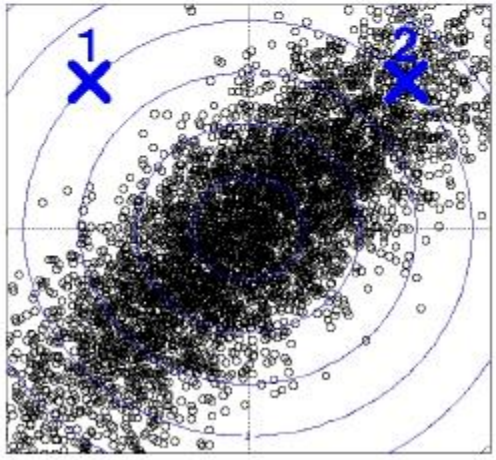
样本点的关键度量指标：距离

- 马氏距离与欧氏距离

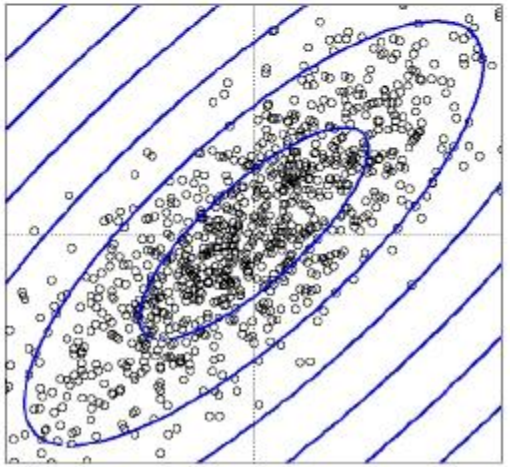




Contour plot of the Euclidian distance to the origin



Contour plot of the Mahalanobis distance to the origin



动态聚类：K-means方法

◆ 算法:

1 选择K个点作为初始质心

2 将每个点指派到最近的质心，形成K个簇（聚类）

3 重新计算每个簇的质心

4 重复2-3直至质心不发生变化

K-means实战代码

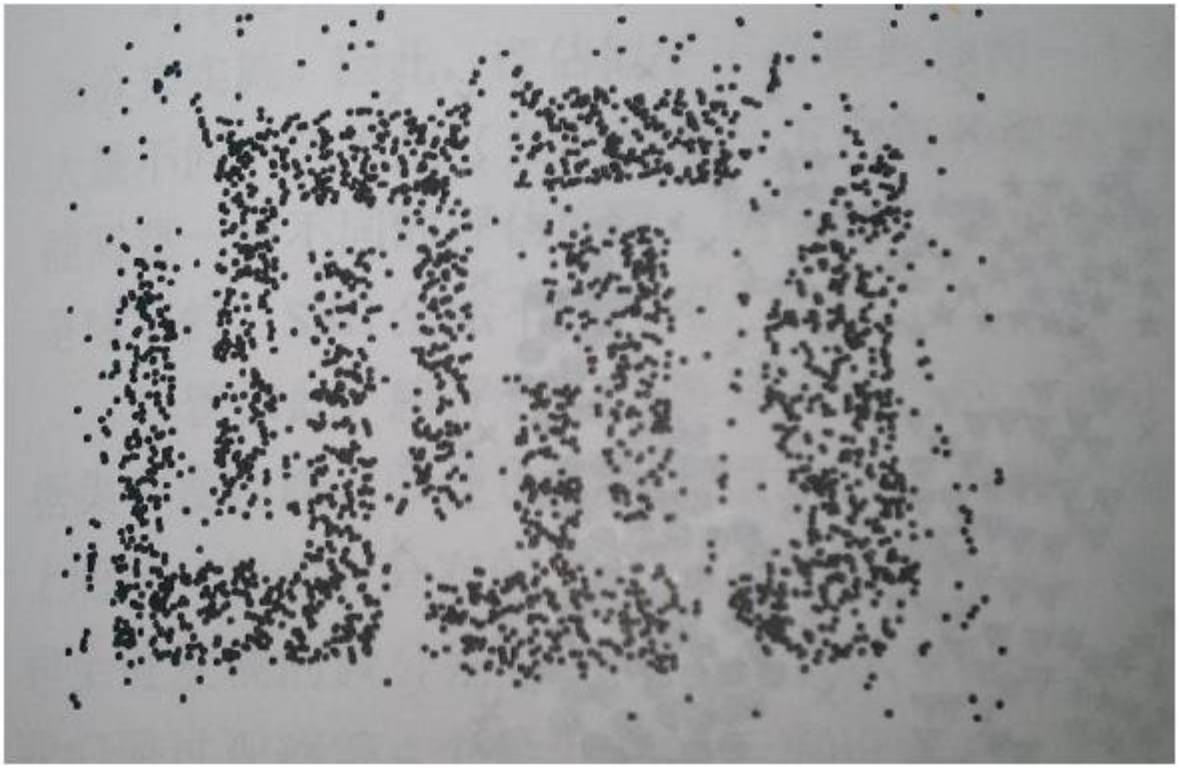
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data[:, :4]  # #表示我们取特征空间中的4个维度
print(X.shape)

estimator = KMeans(n_clusters=3)  # 构造聚类器
estimator.fit(X)  # 聚类
label_pred = estimator.labels_  # 获取聚类标签
# 绘制k-means结果
x0 = X[label_pred == 0]
x1 = X[label_pred == 1]
x2 = X[label_pred == 2]
plt.scatter(x0[:, 0], x0[:, 1], c="red", marker='o', label='label0')
plt.scatter(x1[:, 0], x1[:, 1], c="green", marker='*', label='label1')
plt.scatter(x2[:, 0], x2[:, 1], c="blue", marker='+', label='label2')
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.legend(loc=2)
plt.show()
```

K-means算法的优缺点

- 有效率，而且不容易受初始值选择的影响
- 不能处理非球形的簇
- 不能处理不同尺寸，不同密度的簇
- 离群值可能有较大干扰（因此要先剔除）

基于密度的方法: DBSCAN

- DBSCAN = Density-Based Spatial Clustering of Applications with Noise
- 本算法将具有**足够高密度**的区域划分为簇，并可以发现**任何形状**的聚类



DBSCAN

◆ 算法基本思想

1 指定合适的 r 和 M

2 计算所有的样本点，如果点p的r邻域里有超过M个点，则创建一个以p为核心点的新簇

3 反复寻找这些核心点直接密度可达（之后可能是密度可达）的点，将其加入到相应的簇，对于核心点发生“密度相连”状况的簇，给予合并

4 当没有新的点可以被添加到任何簇时，算法结束

降维

机器学习领域中所谓的降维就是指采用某种映射方法，将原高维空间中的数据点映射到低维度的空间中。降维的本质是学习一个映射函数 f : x->y，其中x是原始数据点的表达，目前最多使用向量表达形式。 y是数据点映射后的低维向量表达，通常y的维度小于x的维度。f可能是显式的或隐式的、线性的或非线性的。

降维技术

- 为何要降维?
  - 使得数据集更易使用
  - 降低算法计算开销
  - 去除噪声
  - 使得结果易懂

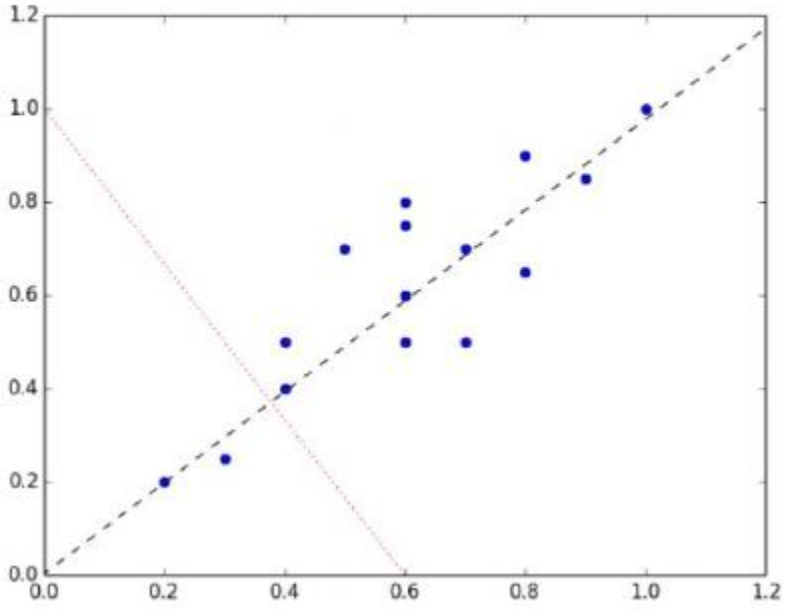
- 主成分分析（PCA）
  - 坐标系转换

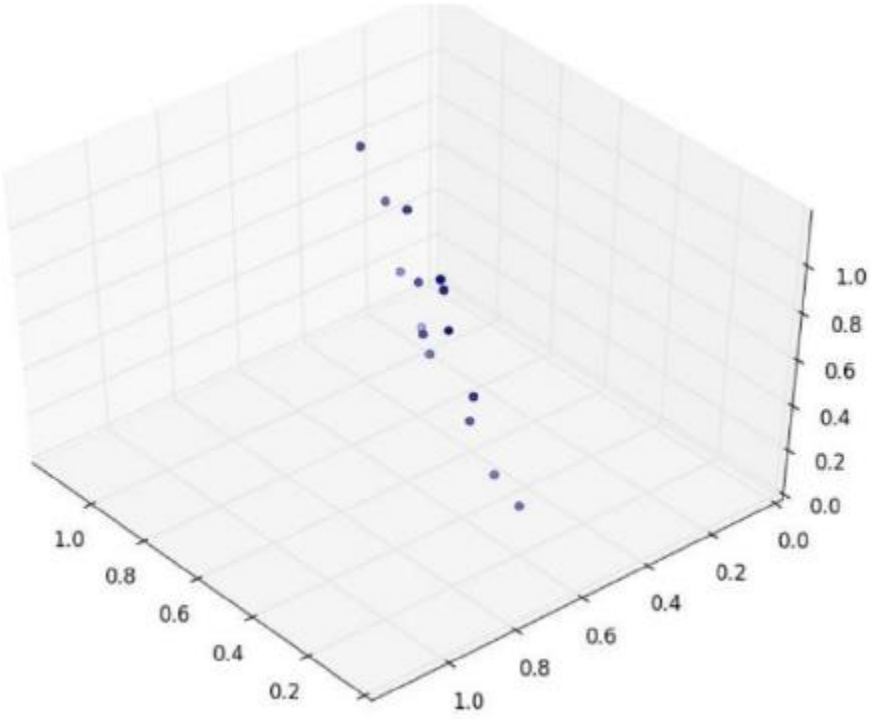
主成分分析

- Pearson于1901年提出，再由Hotelling（1933）加以发展的一种多变量统计方法
- 通过析取主成分显出最大的个别差异，也用来削减回归分析和聚类分析中变量的数目
- 可以使用样本协方差矩阵或相关系数矩阵作为出发点进行分析
- 成分的保留：Kaiser主张（1960）将特征值小于1的成分放弃，只保留特征值大于1的成分
- 如果能用不超过3-5个成分就能解释变异的80%，就算是成功
- 通过对原始变量进行线性组合，得到优化的指标
- 把原先多个指标的计算降维为少量几个经过优化指标的计算（占去绝大部分份额）
- 基本思想：**设法将原先众多具有一定相关性的指标，重新组合为一组新的互相独立的综合指标，并代替原先的指标**

主成分分析

- 优点：降低数据的复杂性，识别最重要的多个特征
- 缺点：不一定需要，且有可能损失有用信息
- 适用数据类型：数值型数据





主成分分析代码

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris

data = load_iris()
y = data.target
X = data.data
pca = PCA(n_components=2)
reduced_X = pca.fit_transform(X)
red_x, red_y = [], []
blue_x, blue_y = [], []
green_x, green_y = [], []
for i in range(len(reduced_X)):
    if y[i] == 0:
        red_x.append(reduced_X[i][0])
        red_y.append(reduced_X[i][1])
    elif y[i] == 1:
        blue_x.append(reduced_X[i][0])
        blue_y.append(reduced_X[i][1])
    else:
        green_x.append(reduced_X[i][0])
        green_y.append(reduced_X[i][1])
plt.scatter(red_x, red_y, c='r', marker='x')
plt.scatter(blue_x, blue_y, c='b', marker='D')
plt.scatter(green_x, green_y, c='g', marker='.')
plt.show()
```

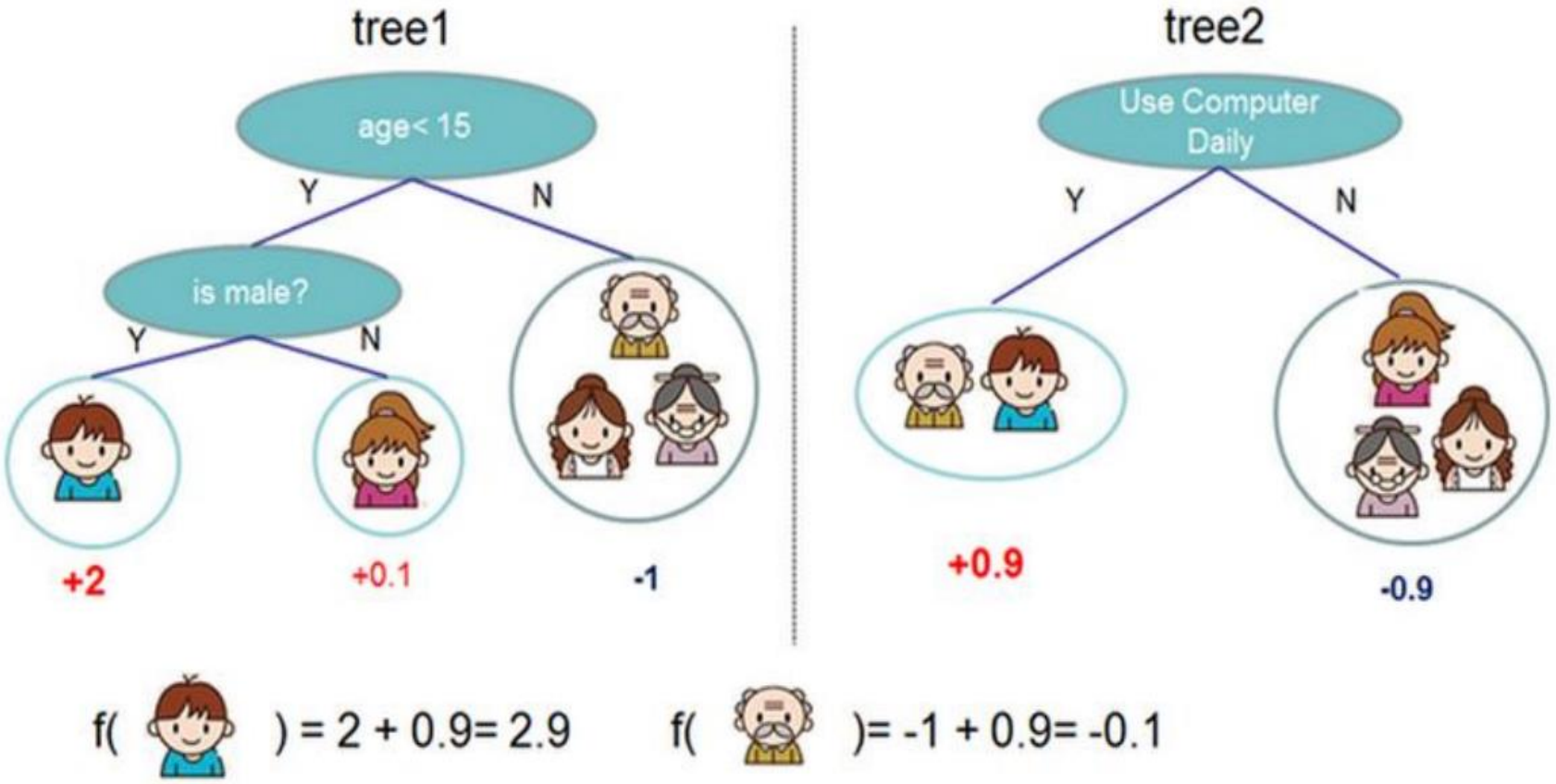
XGBoost

XGBoost是陈天奇等人开发的一个开源机器学习项目，高效地实现了GBDT算法并进行了算法和工程上的许多改进，被广泛应用在Kaggle竞赛及其他许多机器学习竞赛中并取得了不错的成绩。

XGBoost的**核心算法思想:**

1.不断地添加树，不断地进行特征分裂来生长一棵树，每次添加一个树，其实是学习一个新函数**f(x)**，去拟合上次预测的残差。
2.当我们训练完成得到k棵树，我们要预测一个样本的分数，其实就是根据这个样本的特征，在每棵树中会落到对应的一个叶子节点，每个叶子节点就对应一个分数
3.最后只需要将每棵树对应的分数加起来就是该样本的预测值。

举例


tree1
age< 15
Y
N
is male?
Y
N
+2
+0.1
-1
tree2
Use Computer Daily
Y
N
+0.9
-0.9
f( ) = 2 + 0.9= 2.9
f( )= -1 + 0.9= -0.1

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \to \{1, 2, \cdots, T\}$$

树的结构

叶子的向量