层次分析法

3 个 W 原则，是什么？为什么？怎么做？

层次分析法是解决决策以及评价类型的问题，主要的是将问题划分层次，AHP、
你的目标层就是你想要达成的目标，通过准则层的一些指标去判断方案层中那个方案是最佳的选择方案。

层次分析法是什么：按照百度百科的知识我们知道层次分析法，简称 AHP，是指将与决策总是有关的元素分解成目标、准则、方案等层次，在此基础之上进行定性和定量分析的决策方法。因此我们可以明确的知道层次分析法就是一个用来解决决策以及评价类型的题目所使用的方法。

为什么要使用层次分析法：层次分析法有一下优点。

1. 系统性的分析方法、
2. 简洁实用的决策方法
3. 所需定量数据信息较少

怎么使用层次分析法：

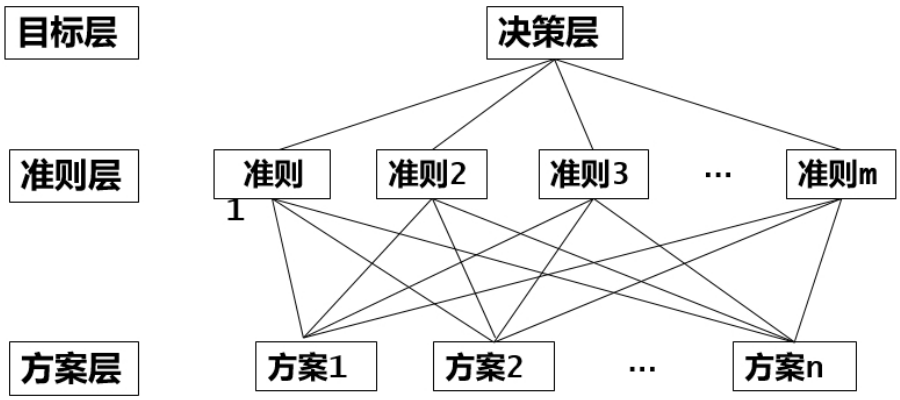
首先层次分析法的使用步骤：

（i）建立递阶层次结构模型；
（ii）构造出各层次中的所有判断矩阵；
（iii）层次单排序及一致性检验；
（iv）层次总排序及一致性检验。

第一步：建立递阶层次结构模型

● 建立层次结构模型

具体层次结构如图9.1所示：



不同层次之间的连线表示其作用关系，同层次因素之间无连线，表示它们互相独立，成为内部独立。上述各层次之间的支配关系不一定是完全的，即可存在某个元素只与下一层次的部分元素有联系，上层元素对下层元素具有分配（或包含）关系，而下层对上层无支配关系，成为递阶层次结构。当某个层次包含因素较多时（如超过9个），可将该层次划分为若干层次。

8

第二步：构造判断矩阵 A，其中 $a_{ji}=\frac{1}{a_{ij}}$

这个判断矩阵的构造一般是构造准则层与方案层这两层的判断矩阵，如果有多层次的话多个因素皆可构造判断矩阵。

判断矩阵的比例值是通过经验判断、多人评审、参考文献等其他途径，来对各元素进行两两比较，来得到判断矩阵，判断矩阵的元素只要符合逻辑和常理，并且满足判断矩阵的一致性。

判断矩阵的特点是：主对角线的值必为 1，并且是呈主对角线以倒数的特点对称。

其中判断矩阵的指标是

关于如何确定 a_{ij} 的值，Saaty 等建议引用数字 1~9 及其倒数作为标度。表 1 列出了 1~9 标度的含义：

表 1　标度的含义

标度	含　　义
1	表示两个因素相比，具有相同重要性
3	表示两个因素相比，前者比后者稍重要
5	表示两个因素相比，前者比后者明显重要
7	表示两个因素相比，前者比后者强烈重要
9	表示两个因素相比，前者比后者极端重要
2，4，6，8	表示上述相邻判断的中间值
倒数	若因素 i 与因素 j 的重要性之比为 a_{ij}，那么因素 j 与因素 i 重要性之比为 $a_{ji}=1/a_{ij}$。

第三步：检验判断矩阵的一致性，这个一致性是通过计算出这个判断矩阵的一致性比例 CR 是否是小于 0.10，若小于 0.10 说明这个判断矩阵的一致性是可以接受的。

➡ **检验判断矩阵 的一致性**

（i）计算一致性指标*CI*: $CI = \dfrac{\lambda_{\max} - n}{n-1}$

（ii）查找相应的平均随机一致性指标*RI*，对 $n = 1, 2, \cdots, 9$ ***RI* 的值如表2 所示：**

表9.2 平均随机一致性指标 *RI* 值

阶数	1	2	3	4	5	6	7	8	9
RI	0	0	0.58	0.9	1.12	1.24	1.32	1.41	1.45

（iii）计算一致性比例*CR*: $CR = \dfrac{CI}{RI}$

当*CR* < 0.10时，认为判断矩阵的一致性是可以接受的，否则应对判断矩阵作适当修正。

20

第四步：层次总排序及一致性检验

这个也是得出最后的结果的一种合成表，包含着准则层的判断矩阵的权重以及方案层关于每一准则层的判断矩阵的权重值，以及最后对应乘积求和的最终权重得出决策。

层 A / 层 B	A_1	A_2	$\cdots$	A_m	B 层总排序权值
	a_1	a_2	$\cdots$	a_m	
B_1	b_{11}	b_{12}	$\cdots$	b_{1m}	$\sum_{j=1}^{m} b_{1j}a_j$
B_2	b_{21}	b_{22}	$\cdots$	b_{2m}	$\sum_{j=1}^{m} b_{2j}a_j$
$\vdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\vdots$
B_n	b_{n1}	b_{n2}	$\cdots$	b_{nm}	$\sum_{j=1}^{m} b_{nj}a_j$

Python 实现：

```
import numpy as np

class AHP: # 构造一个关于层次分析法求权重值的类
```

```
    def __init__(self, array): # 这个是这个类的构造函数
        ## 记录矩阵相关信息
        self.array = array
        ## 记录矩阵大小，行数即为阶数
        self.n = array.shape[0]  # 在 Numpy 库中，array.shape 函数的作用是可以直接获取矩阵的行数和列数，并且这个 shape[0]获取行数，shape[1]获取列数。
        # 初始化 RI 值，用于一致性检验
        self.RI_list = [0, 0, 0.52, 0.89, 1.12, 1.26, 1.36, 1.41, 1.46, 1.49, 1.52, 1.54, 1.56, 1.58,
                        1.59] # 定义一个列表存储 RI 数据
        # 矩阵的特征值和特征向量
        # numpy.linalg 模块包含线性代数的函数。使用这个模块，可以计算逆矩阵、求特征值、解线性方程组以及求解行列式等。
        self.eig_val, self.eig_vector = np.linalg.eig(self.array)
        # 矩阵的最大特征值
        self.max_eig_val = np.max(self.eig_val)
        # 矩阵最大特征值对应的特征向量
        self.max_eig_vector = self.eig_vector[:, np.argmax(self.eig_val)].real
        # 矩阵的一致性指标 CI
        self.CI_val = (self.max_eig_val - self.n) / (self.n - 1)
        # 矩阵的一致性比例 CR
        self.CR_val = self.CI_val / (self.RI_list[self.n - 1])

    """
    一致性判断
    """

    def test_consist(self):
        # 打印矩阵的一致性指标 CI 和一致性比例 CR
        print("判断矩阵的 CI 值为：" + str(self.CI_val))
        print("判断矩阵的 CR 值为：" + str(self.CR_val))
        # 进行一致性检验判断
        if self.n == 2:  # 当只有两个子因素的情况
            print("仅包含两个子因素，不存在一致性问题")
        else:
            if self.CR_val < 0.1:  # CR 值小于 0.1，可以通过一致性检验
                print("判断矩阵的 CR 值为" + str(self.CR_val) + ",通过一致性检验")
                return True
            else:  # CR 值大于 0.1，一致性检验不通过
                print("判断矩阵的 CR 值为" + str(self.CR_val) + "未通过一致性检验")
                return False

    """
```

```
    算术平均法求权重
    """

    def cal_weight_by_arithmetic_method(self):
        # 求矩阵的每列的和
        col_sum = np.sum(self.array, axis=0)
        # 将判断矩阵按照列归一化
        array_normed = self.array / col_sum
        # 计算权重向量
        array_weight = np.sum(array_normed, axis=1) / self.n
        # 打印权重向量
        # print("算术平均法计算得到的权重向量为：\n", array_weight)
        # 返回权重向量的值
        return array_weight

    """
    几何平均法求权重
    """

    def cal_weight__by_geometric_method(self):
        # 求矩阵的每列的积
        col_product = np.product(self.array, axis=0)
        # 将得到的积向量的每个分量进行开 n 次方
        array_power = np.power(col_product, 1 / self.n)
        # 将列向量归一化
        array_weight = array_power / np.sum(array_power)
        # 打印权重向量
        print("几何平均法计算得到的权重向量为：\n", array_weight)
        # 返回权重向量的值
        return array_weight

    """
    特征值法求权重
    """

    def cal_weight__by_eigenvalue_method(self):
        # 将矩阵最大特征值对应的特征向量进行归一化处理就得到了权重
        array_weight = self.max_eig_vector / np.sum(self.max_eig_vector)
        # 打印权重向量
        print("特征值法计算得到的权重向量为：\n", array_weight)
        # 返回权重向量的值
        return array_weight
```

```
if __name__ == "__main__":
    # 给出判断矩阵
    e = np.array([[1, 2, 7, 5, 5], [1 / 2, 1, 4, 3, 3], [1 / 7, 1 / 4, 1, 1 / 2, 1
/ 3], [1 / 5, 1 / 3, 2, 1, 1], [1 / 5, 1 / 3, 3, 1, 1]]) # 五维准则层
    a = np.array([[1, 1 / 3, 1 / 8], [3, 1, 1 / 3], [8, 3, 1]]) # 每一个准则层的方
案层
    b = np.array([[1, 2, 5], [1 / 2, 1, 2], [1 / 5, 1 / 2, 1]])
    c = np.array([[1, 1, 3], [1, 1, 3], [1 / 3, 1 / 3, 1]])
    d = np.array([[1, 3, 4], [1 / 3, 1, 1], [1 / 4, 1, 1]])
    f = np.array([[1, 4, 1 / 2], [1 / 4, 1, 1 / 4], [2, 4, 1]])

    # 算术平均法求权重
    e = AHP(e).cal_weight_by_arithmetic_method()
    a = AHP(a).cal_weight_by_arithmetic_method()
    b = AHP(b).cal_weight_by_arithmetic_method()
    c = AHP(c).cal_weight_by_arithmetic_method()
    d = AHP(d).cal_weight_by_arithmetic_method()
    f = AHP(f).cal_weight_by_arithmetic_method()

    try:
        res = np.array([a, b, c, d, f])
        ret = (np.transpose(res) * e).sum(axis=1)
        print(ret)
    except TypeError:
        print('数据有误，可能不满足一致性，请进行修改')
```