熵权法

> 具有良好的客观性；
>
> 目标是求权重。

In [ ]:
```
import pandas as pd

import numpy as np
```

正向化指标

- 极小型指标

  $max - x$ 或 $\frac{1}{x}(x > 0)$

In [ ]:
```
def Min2Max(X):
    """
    极小型转极大型
    """
    X=[max(X) - x_i for x_i in X]
    return X
```

- 中间型指标

  $M = max\{|x_i - x_{best}|\}$

  $\widetilde{x_i} = 1 - \frac{|x_i - X_best|}{M}$

In [ ]:
```
def Mid2Max(X, best):
    """
    中间型转极大型
    """
    M = max([abs(x_i - best) for x_i in X])
    X = [1 - abs(x_i - best) / M for x_i in X]
    return X
```

- 区间型指标

  $M = max\{a - min\{x_i\}, max\{x_i\} - b\}$

$$\widetilde{x_i} = \begin{cases} 1 - \frac{a - x_i}{M} & x_i < a \\ 1 & a < x_i < b \\ 1 - \frac{x_i - b}{M} & x_i > b \end{cases}$$

In [ ]:
```python
def Int2Max(X, a, b):
    """
    区间型转极大型
    """
    M = max([a - min(X)] + [max(X) - b])
    for x_i in X:
        if x_i < a:
            X[X.index(x_i)] = 1 - (a - x_i) / M
        elif x_i > b:
            X[X.index(x_i)] = 1 - (x_i - b) / M
        else:
            X[X.index(x_i)] = 1
    return X
```

➢第一步：正向化处理

区间型转极大型：数值不要太大也不要太小，落在某个区间最好，如人体温度值落在36～37℃最好.

设对于一组区间型指标$\{x_i\}$其最佳区间为$[a,b]$那么可以这样正向化：

先算最值到边界的最大距离$M=\max\{a-\min\{x_i\},\max\{x_i\}-b\}$，再令各元素$\tilde{x}_i=\begin{cases}1-\frac{a-x_i}{M} & ,x_i<a\\ 1 & ,a\le x_i\le b\\ 1-\frac{x_i-b}{M} & ,x_i>b\end{cases}$

体温	正向化后的体温
35.2	0.2
35.8	0.8
36.5	1
37.2	0.8
38.0	0

$a=36,b=37,M=\max\{36-35.2,38.0-37\}=1$

In [ ]:
```python
data = pd.read_excel("第 4 讲 - 熵权法.xlsx")
```

例题表格如下：

In [ ]:
```python
data
```

Out[ ]:

	Unnamed: 0	年龄	身高	长相	学历(成绩)	收入
0	大师兄	24	180	60	90	5000
1	小A	23	175	90	85	10000
2	小B	23	170	95	92	8000
3	小C	24	185	81	100	5500
4	小D	25	190	79	60	2000

In [ ]:
```python
# 数据预处理

name = list(data["Unnamed: 0"])
age = list(data["年龄"])
height = list(data["身高"])
appearance = list(data["长相"])
```

```
grade = list(data["学历(成绩)"])
income = list(data["收入"])
```

In [ ]:
```
# 正向化处理

age = Min2Max(age)

height = Mid2Max(height, 180)

appearance = Int2Max(appearance, 80, 90)
```

In [ ]:
```
age, height, appearance, grade, income
```

Out[ ]:
```
([1, 2, 2, 1, 0],
 [1.0, 0.5, 0.0, 0.5, 0.0],
 [0.0, 1, 0.75, 1, 0.95],
 [90, 85, 92, 100, 60],
 [5000, 10000, 8000, 5500, 2000])
```

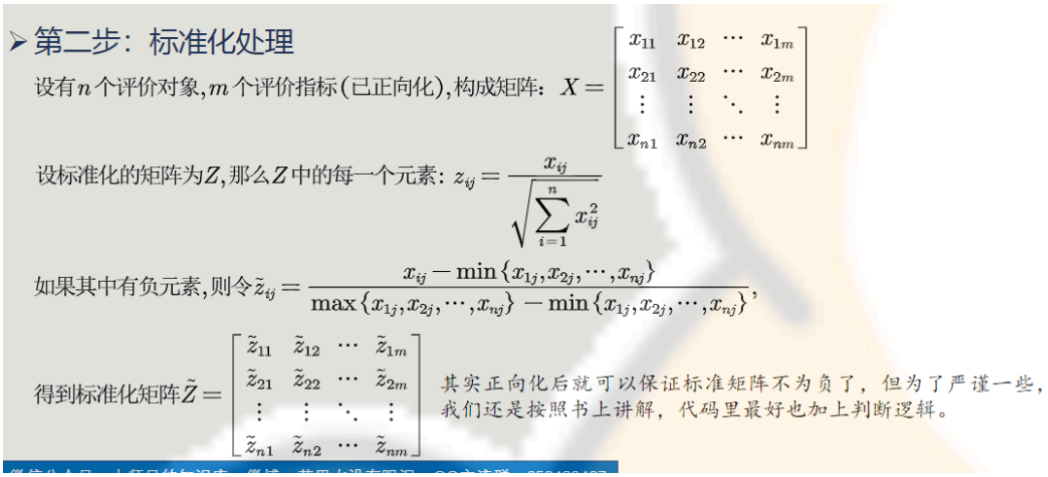
In [ ]:
```
# 输出正向化后的矩阵

# 其实在这篇代码中，这一段可以删去，因为后面标准化我直接用的前面的向量

X_row = [age, height, appearance, grade, income]
x_row = np.array(X_row)
x_row = x_row.T
print("X = \n", x_row)
```

```
X =
 [[1.0e+00 1.0e+00 0.0e+00 9.0e+01 5.0e+03]
 [2.0e+00 5.0e-01 1.0e+00 8.5e+01 1.0e+04]
 [2.0e+00 0.0e+00 7.5e-01 9.2e+01 8.0e+03]
 [1.0e+00 5.0e-01 1.0e+00 1.0e+02 5.5e+03]
 [0.0e+00 0.0e+00 9.5e-01 6.0e+01 2.0e+03]]
```

标准化处理



> 标准化的意义，是为了消除量纲的影响，使不同量纲的指标处于同一数量级，从而能够进行比较。

In [ ]:
```
def calculate_Z_vector(x_row):
    """
    计算标准化后的矩阵

    参数是一维列向量，也就是 X_row 的每一个元素，例如 age, height, appearance, gr

    """
    x_row = [x_i / (sum(np.square(x_row)) ** (1 / 2)) for x_i in x_row]
    return x_row
```

In [ ]:
```
# 计算标准化后的矩阵

# 不过我没有判断非负性

Z_row = [calculate_Z_vector(X_row[i]) for i in range(len(X_row))]
Z = np.array(Z_row)
Z = Z.T
print("Z = \n", Z)
```

```
Z =
 [[0.31622777 0.81649658 0.         0.46544737 0.33463724]
 [0.63245553 0.40824829 0.53721531 0.43958919 0.66927448]
 [0.63245553 0.         0.40291148 0.47579065 0.53541959]
 [0.31622777 0.40824829 0.53721531 0.51716375 0.36810096]
 [0.         0.         0.51035454 0.31029825 0.1338549 ]]
```

计算信息熵和熵权

➢计算各个指标下每个样本的比重

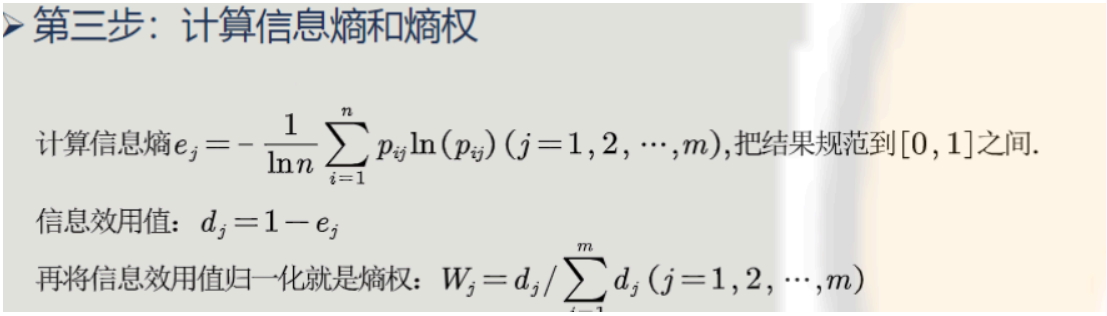
然后计算比重矩阵P,其中P中每一个元素p_{ij}的计算公式：$p_{ij}=\frac{\tilde{z}_{ij}}{\sum_{i=1}^{n}\tilde{z}_{ij}}$,容易验证:$\sum_{i=1}^{n}p_{ij}=1$.

就是算每个指标在这一列所占的比重.我们把这个比重看成概率.

In [ ]:
```
def calculate_P_vector(z_row):
    """
    计算比重矩阵
    """
    z_row = [z_i / sum(z_row) for z_i in z_row]
    return z_row
```

In [ ]:
```
# 输出比重矩阵
P_row = [calculate_P_vector(X_row[i]) for i in range(len(X_row))]
P = np.array(P_row)
P = P.T
print("P = \n", P)
```

```
P =
 [[0.16666667 0.5        0.         0.21077283 0.16393443]
 [0.33333333 0.25       0.27027027 0.19906323 0.32786885]
 [0.33333333 0.         0.2027027  0.21545667 0.26229508]
 [0.16666667 0.25       0.27027027 0.23419204 0.18032787]
 [0.         0.         0.25675676 0.14051522 0.06557377]]
```



$I(x) = -ln(p(x))$

In [ ]:
```
def calculate_entropy_vector(P):
    """
    计算规范后的信息熵。
    """
    n, m = P.shape
    entropy_vector = np.zeros(m)

    for j in range(m):
        entropy = 0.0
        for i in range(n):
            if P[i, j] > 0:
                entropy += P[i, j] * np.log(P[i, j])
        entropy_vector[j] = -entropy / np.log(n)

    return entropy_vector
```

$e_j = -\frac{1}{lnn}\sum_{i=1}^{n}p_{ij}ln(p(x_{ij}))(j=1,2,\ldots,m)$

In [ ]:
```
# 将 0 替换成 0.00001

zero = 1e-5

P[P == 0] = zero
```

In [ ]:
```
e_j = calculate_entropy_vector(P)  # 计算信息熵

d_j = 1 - e_j  # 信息效用值

d_j
```

Out[ ]:
```
array([0.17376342, 0.3538421 , 0.14260018, 0.00840138, 0.06759664])
```

In [ ]:
```
W_j = d_j / sum(d_j)  # 熵权
```

In [ ]:
```
# 输出结果

df_head = list(data.columns)

df_head[0] = ""

df_result = ["权值"] + list(W_j)

df = pd.DataFrame(df_result)

df = df.T
```

```
df.columns = df_head

df
```

Out[ ]:

		年龄	身高	长相	学历(成绩)	收入
0	权值	0.232863	0.47419	0.191101	0.011259	0.090587