Matlab代码注释

R = []; % 创建一个空矩阵 R

这行代码创建了一个空矩阵 **R**。你需要将你的数据导入R

[rows, cols] = size(R); % 获取矩阵 R 的大小（行数和列数）

这行代码计算矩阵 **R** 的行数（对象数）和列数（指标数）。

k = 1 / log(rows); % 计算 k

变量 **k** 是根据行数计算的，将在后续计算中使用。

Rmin = min(R); Rmax = max(R); A = max(R) - min(R); y = R - repmat(Rmin, rows, 1);

这些代码计算了矩阵 **R** 每列的最小值（**Rmin**）、最大值（**Rmax**）和振幅（**A**），同时计算了新的矩阵 **y**，该矩阵是相对于最小值进行归一化的版本。

for j = 1 : size(y, 2) y(:, j) = y(:, j) / A(j); end

这个循环遍历 **y** 的每一列，并将每列的值归一化为介于 0 和 1 之间。

S = sum(y, 1);

这行代码计算了矩阵 **y** 每列的和。

Y = zeros(rows, cols); for i = 1 : size(Y, 2) Y(:, i) = y(:, i) / S(i); end

在这里，矩阵 **Y** 是通过将 **y** 的每个元素除以相应列的和来计算的。这将使得每列的值归一化，总和为 1。

lnYij = zeros(rows, cols); for i = 1:rows for j = 1:cols if Y(i, j) == 0 lnYij(i, j) = 0; else lnYij(i, j) = log(Y(i, j)); end end end

这个循环计算矩阵 **Y** 中每个元素的自然对数，同时处理元素为零的情况。

ej = -k \* (sum(Y .\* lnYij, 1));

这行代码计算了每个指标的熵

weights = (1 - ej) / (cols - sum(ej));

在这里，基于熵值，计算了每个指标的权重。

F = zeros(rows, cols); for k = 1 : size(R, 2) F(:, k) = weights(k) \* y(:, k); end

这个循环计算了综合评价的加权和。

F = sum(F, 2); % F 即为综合评价分数

最后，计算了每个对象的加权和，得到了综合评价分数 **F**。

总的来说，这段代码实现了熵权法的计算步骤，包括数据的归一化、熵的计算、权重的确定以及利用这些权重进行综合评价。最终的矩阵 **F** 包含了基于给定指标的对象的综合评价分数。

补充：

这行代码进行了矩阵的运算，计算了归一化后的矩阵 **y**。让我们解释这一步骤。

首先，让我们回顾一下之前的几步，特别是归一化过程：

1. **Rmin = min(R);**：计算了矩阵 **R** 每列的最小值，得到了一个包含最小值的行向量 **Rmin**。
2. **Rmax = max(R);**：计算了矩阵 **R** 每列的最大值，得到了一个包含最大值的行向量 **Rmax**。
3. **A = max(R) - min(R);**：计算了矩阵 **R** 每列的振幅，得到了一个包含振幅的行向量 **A**。

现在，针对归一化的计算：

y = R - repmat(Rmin, rows, 1);

这行代码使用 **repmat** 函数将 **Rmin** 行向量复制成 **rows** 行，每行都是 **Rmin**。然后，它从每列中减去相应的 **Rmin**，实现了对矩阵 **R** 的归一化处理。这样做的目的是使得每个指标的最小值变为0。这个过程有助于在后续步骤中更好地进行归一化处理，确保所有指标都在相似的范围内。

**python实现**

**3.1 数据准备**

为方便读者测试，这边手动生成一段数据作为示例。

将指标1，指标2，指标3，指标4，合并编制为一个“综合指标”。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | import pandas as pd  import numpy as np    # 1. 初始数据 假设指标4是负向指标，其余三个为正向指标  df1 = pd.DataFrame({'指标1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],                      '指标2': [2, 4, 6, 8, 10, 2, 4, 6, 8, 10],                      '指标3': [1, 2, 1, 3, 2, 1, 3, 2, 3, 1],                      '指标4': [3, 1, 2, 3, 5, 8, 7, 8, 8, 9]                     })  print(df1) |

数据为DataFrame格式，效果展示如下：



**3.2 数据预处理**

然后是数据预处理部分，这里定义一个泛用性较强的标准化处理函数，

该函数对于正向指标和负向指标（越大越好的指标和越小越好的指标），可以分别进行不同的处理。

（处理逻辑通过代码可以很容易看出）

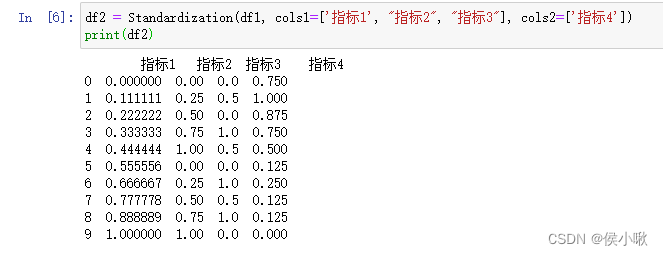
同时该函数也可以兼容只进行其中一种处理的情景。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | # 2.数据预处理 定义标准化处理函数  def Standardization(data,cols1=None, cols2=None):      """      :param data:目标数据      :param cols1: 需要处理的正向指标列名列表，类型为列表或None [col1, col2, col3]      :param cols2: 需要处理的负向指标列名列表，类型为列表或None [col1, col2, col3]      :return: 输出处理结果      """      if cols1 == None and cols2 == None:          return data      elif cols1 != None and cols2 == None:          return (data[cols1] - data[cols1].min())/(data[cols1].max()-data[cols1].min())      elif cols1 == None and cols2 != None:          return (data[cols2].max - data[cols2])/(data[cols2].max()-data[cols2].min())      else:          a = (data[cols1] - data[cols1].min())/(data[cols1].max()-data[cols1].min())          b = (data[cols2].max() - data[cols2])/(data[cols2].max()-data[cols2].min())          return pd.concat([a, b], axis=1) |

调用函数，进行标准化处理：

|  |  |
| --- | --- |
| 1  2 | df2 = Standardization(df1, cols1=['指标1', "指标2", "指标3"], cols2=['指标4'])  print(df2) |

处理结果如下：



**3.3 熵值、权重计算**

然后定义一个通过熵值计算权重，及样本评分的函数。

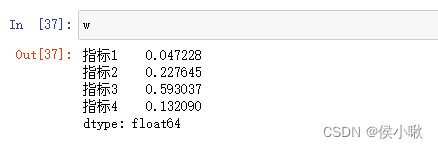
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | def Weightfun(data):      """      :param data: 预处理好的数据      :return: 输出权重。      """      K = 1/np.log(len(data))      e = -K\*np.sum(data\*np.log(data))      d = 1-e      w = d/d.sum()      return w |

该函数的返回值有两个，w是权重，score是评分

调用函数，计算出各个指标的权重：

|  |  |
| --- | --- |
| 1  2 | w = Weightfun(df2)  print(w) |

各个指标权重如下图所示：



**3.4 编制综合评价指标**

直接将DataFrame格式的数据与求出的权重相乘，并求和，即得到通过熵值法编制出的综合指标，代码及结果如下图所示：

|  |  |
| --- | --- |
| 1  2 | df3= df2 \* w  df3['综合指标'] = df3.sum(axis=1) |